

Software

IDC DOCUMENTATION

Archiving Subsystem



**Approved for public release;
distribution unlimited**

Notice

This document was published May 2001 by the Monitoring Systems Operation of Science Applications International Corporation (SAIC) as part of the International Data Centre (IDC) Documentation. Every effort was made to ensure that the information in this document was accurate at the time of publication. However, information is subject to change.

Contributors

Anna Katherine Gault, Science Applications International Corporation
David Salzberg, Science Applications International Corporation

Trademarks

ORACLE is a registered trademark of Oracle Corporation.
Solaris is a registered trademark of Sun Microsystems.
SPARC is a registered trademark of Sun Microsystems.
SQL *Plus is a registered trademark of Oracle Corporation.
Sun is a registered trademark of Sun Microsystems.
UNIX is a registered trademark of UNIX System Labs, Inc.

Ordering Information

The ordering number for this document is SAIC-01/3013.

This document is cited within other IDC documents as [IDC7.5.1].

Archiving Subsystem

CONTENTS

About this Document	i
■ PURPOSE	ii
■ SCOPE	ii
■ AUDIENCE	ii
■ RELATED INFORMATION	iii
■ USING THIS DOCUMENT	iii
Conventions	iv
Chapter 1: Overview	1
■ INTRODUCTION	2
■ FUNCTIONALITY	5
■ IDENTIFICATION	5
■ STATUS OF DEVELOPMENT	5
■ BACKGROUND AND HISTORY	6
■ OPERATING ENVIRONMENT	6
Hardware	6
Commercial-Off-The-Shelf Software	7
Chapter 2: Architectural Design	9
■ CONCEPTUAL DESIGN	10
■ DESIGN ISSUES	11
Programming Language	11
Global Libraries	12
Database	12
Interprocess Communication (IPC)	12
Filesystem	12
Design Model	12

Database Schema Overview	13
■ FUNCTIONAL DESCRIPTION	14
Create Intervals	15
Queue Intervals	16
Run Interval	17
Monitoring	17
■ INTERFACE DESIGN	19
Interface with Other IDC Systems	19
Interface with Internal Users	20
Interface with External Users	20
Interface with Operators	20
Chapter 3: Detailed Design	21
■ DATA FLOW MODEL	22
External Data Flow	22
Internal Data Flow	23
Flow of Internal Data Exchange (Archive Protocol)	26
■ PROCESSING UNITS	31
Archive	31
ManageInterval	34
GetData	36
MergeData	37
ReadWriteData	39
MSwriter	41
■ DATABASE DESCRIPTION	43
Database Design	43
Database Schema	45
Chapter 4: Requirements	49
■ INTRODUCTION	50
■ GENERAL REQUIREMENTS	50
■ FUNCTIONAL REQUIREMENTS	51
Generic Functional Requirements	51

User Interface	51
Exception Handling and Recovery Procedures	52
■ SYSTEM REQUIREMENTS	53
■ REQUIREMENTS TRACEABILITY	53
References	59
Appendix: Defining Fileproducts	A1
Glossary	G1
Index	I1

Archiving Subsystem

FIGURES

FIGURE 1.	IDC SOFTWARE CONFIGURATION HIERARCHY	3
FIGURE 2.	RELATIONSHIP OF ARCHIVING SUBSYSTEM TO CONTINUOUS DATA AND MESSAGE SUBSYSTEMS	4
FIGURE 3.	REPRESENTATIVE HARDWARE CONFIGURATION FOR ARCHIVING SUBSYSTEM	7
FIGURE 4.	CONCEPTUAL MODEL OF ARCHIVING SUBSYSTEM	11
FIGURE 5.	ARCHIVING SUBSYSTEM INTERVAL PROCESSING AND STATUS	16
FIGURE 6.	PROCESSING FLOW FOR RUN INTERVAL	18
FIGURE 7.	DATA FLOW OF ARCHIVING SUBSYSTEM	23
FIGURE 8.	DATA FLOW OF ARCHIVE PROCESSING	25
FIGURE 9.	DATA FLOW OF PROTOCOL EXCHANGE BETWEEN ARCHIVE AND MSWRITER	30
FIGURE 10.	RELATIONSHIPS BETWEEN TABLES USED FOR ARCHIVING SUBSYSTEM	45

Archiving Subsystem

TABLES

TABLE I:	DATA FLOW SYMBOLS	iv
TABLE II:	ENTITY-RELATIONSHIP SYMBOLS	v
TABLE III:	TYPOGRAPHICAL CONVENTIONS	vi
TABLE 1:	SUPPORTED CONFIGURATIONS OF ARCHIVING SUBSYSTEM	11
TABLE 2:	ARCHIVING SUBSYSTEM DATABASE TABLE USE	13
TABLE 3:	SUPPORTED DATA TYPES AND DATABASE TABLES	15
TABLE 4:	STRUCTURE OF COMMUNICATIONS FRAME	27
TABLE 5:	DATA FRAME TYPES FOR COMMUNICATIONS FRAME	28
TABLE 6:	VALID STATUS VALUES FOR COMMUNICATION FRAMES	28
TABLE 7:	STRUCTURE OF ARCHIVING SUBSYSTEM TIMESTAMP FRAME	28
TABLE 8:	STRUCTURE OF FILE AND DATA SEGMENT HEADER FRAMES	29
TABLE 9:	DATABASE USAGE BY ARCHIVING SUBSYSTEM	46
TABLE 10:	TRACEABILITY OF GENERAL REQUIREMENTS	53
TABLE 11:	TRACEABILITY OF FUNCTIONAL REQUIREMENTS: GENERIC FUNCTIONAL REQUIREMENTS	54
TABLE 12:	TRACEABILITY OF FUNCTIONAL REQUIREMENTS: USER INTERFACE	55
TABLE 13:	TRACEABILITY OF FUNCTIONAL REQUIREMENTS: EXCEPTION HANDLING AND RECOVERY PROCEDURES	56
TABLE 14:	TRACEABILITY OF FUNCTIONAL REQUIREMENTS: SYSTEM REQUIREMENTS	57

About this Document

This chapter describes the organization and content of the document and includes the following topics:

- Purpose
- Scope
- Audience
- Related Information
- Using this Document

About this Document

PURPOSE

This document describes the design and requirements of the Archiving Subsystem software of the International Data Centre (IDC). The software is part of the Data Archiving computer software component (CSC) of the Data Management Computer Software Configuration Item (CSCI). This document provides a basis for implementing, supporting, and testing the software.

SCOPE

The Archiving Subsystem software is identified as follows:

Title: Archiving Subsystem

This document describes the architectural and detailed design of the software including its functionality, components, data structures, high-level interfaces, method of execution, and underlying hardware. Additionally, this document specifies the requirements of the software and its components. This information is modeled on the Data Item Description for Software Design Descriptions [DOD94a] and Software Requirements Specification [DOD94b]. This document does not describe the modules under development.

AUDIENCE

This document is intended for all engineering and management staff concerned with the design and requirements of all IDC software in general and of the Archiving Subsystem in particular. The detailed descriptions are intended for programmers who will be developing, testing, or maintaining the Archiving Subsystem.

RELATED INFORMATION

The following document complements this document:

- *Database Schema* [IDC5.1.1Rev2]

See “References” on page 59 for a list of documents that supplement this document. The following UNIX manual (man) pages apply to the existing Archiving Subsystem software:

- *Archive*
- *Mswriter*

USING THIS DOCUMENT

This document is part of the overall documentation architecture for the IDC. It is part of the Software category, which describes the design of the software. This document is organized as follows:

- Chapter 1: Overview
This chapter provides a high-level view of the Archiving Subsystem, including its functionality, components, background, status of development, and current operating environment.
- Chapter 2: Architectural Design
This chapter describes the architectural design of the Archiving Subsystem, including its conceptual design, design decisions, functions, and interface design.
- Chapter 3: Detailed Design
This chapter describes the detailed design of the Archiving Subsystem including its data flow, software units, and database design.
- Chapter 4: Requirements
This chapter describes the general, functional, and system requirements for the Archiving Subsystem. Traceability tables define how these requirements are met.

▼ About this Document

- References
This section lists the sources cited in this document.
- Appendix: Defining Fileproducts
This appendix describes the procedure for defining fileproducts.
- Glossary
This section defines the terms, abbreviations, and acronyms used in this document.
- Index
This section lists topics and features provided in the document along with page numbers for reference.

Conventions

This document uses a variety of conventions, which are described in the following tables. Table I shows the conventions for data flow diagrams. Table II shows the conventions for entity-relationship (E-R) diagrams. Table III lists typographical conventions.

TABLE I: DATA FLOW SYMBOLS

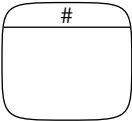


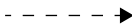

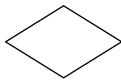
Description	Symbol ¹
process	
external source or sink of data	
data store D = disk store Db = database store MS = mass store	

TABLE I: DATA FLOW SYMBOLS (CONTINUED)

Description	Symbol ¹
control flow	
data flow	
decision	

1. Symbols in this table are based on Gane-Sarson conventions [Gan79].

TABLE II: ENTITY-RELATIONSHIP SYMBOLS




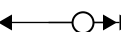






Description	Symbol			
One A maps to one B.	A  B			
One A maps to zero or one B.	A  B			
One A maps to many Bs.	A  B			
One A maps to zero or many Bs.	A  B			
database table	<table><tr><td>tablename</td></tr><tr><td> <i>primary key</i>  <i>foreign key</i></td></tr><tr><td><i>attribute 1</i> <i>attribute 2</i> . . . <i>attribute n</i></td></tr></table>	tablename	 <i>primary key</i>  <i>foreign key</i>	<i>attribute 1</i> <i>attribute 2</i> . . . <i>attribute n</i>
tablename				
 <i>primary key</i>  <i>foreign key</i>				
<i>attribute 1</i> <i>attribute 2</i> . . . <i>attribute n</i>				

TABLE III: TYPOGRAPHICAL CONVENTIONS

Element	Font	Example
database table	bold	wfdisc
database table and attribute, when written in the dot notation		interval.state
database attributes	<i>italics</i>	<i>state</i>
processes, software units, and libraries		<i>Archive</i>
user-defined arguments and variables used in parameter (par) files or program command lines		<i>xtype = C</i>
titles of documents		<i>Database Schema</i>
computer code and output text that should be typed in exactly as shown	courier	Arch_opsdb failure edit-filter-dialog
attribute values		QUEUED

Chapter 1: Overview

This chapter provides a general overview of the Archiving Subsystem software and includes the following topics:

- Introduction
- Functionality
- Identification
- Status of Development
- Background and History
- Operating Environment

Chapter 1: Overview

INTRODUCTION

The software of the IDC acquires time-series and radionuclide data from stations of the International Monitoring System (IMS) and other locations. These data are passed through a number of automatic and interactive analysis stages, which culminate in the estimation of location and in the origin time of events (earthquakes, volcanic eruptions, and so on) in the earth, including its oceans and atmosphere. The results of the analysis are distributed to States Parties and other users by various means. Approximately one million lines of developmental software are spread across six CSCIs of the software architecture. One additional CSCI is devoted to run-time data of the software. Figure 1 shows the logical organization of the IDC software. The Data Management CSCI receives and archives data through the following CSCs:

- Data Archiving
This software migrates data from the operations database to the archive database. It also archives data on the mass storage device and fills the archive of waveform segments associated to events.
- Database Libraries
This software consists of libraries and include files needed for database access.
- Database Tools
This software consists of utilities used in connection with database access.
- Configuration Management
This software consists of scripts that facilitate recurring tasks in configuration management.

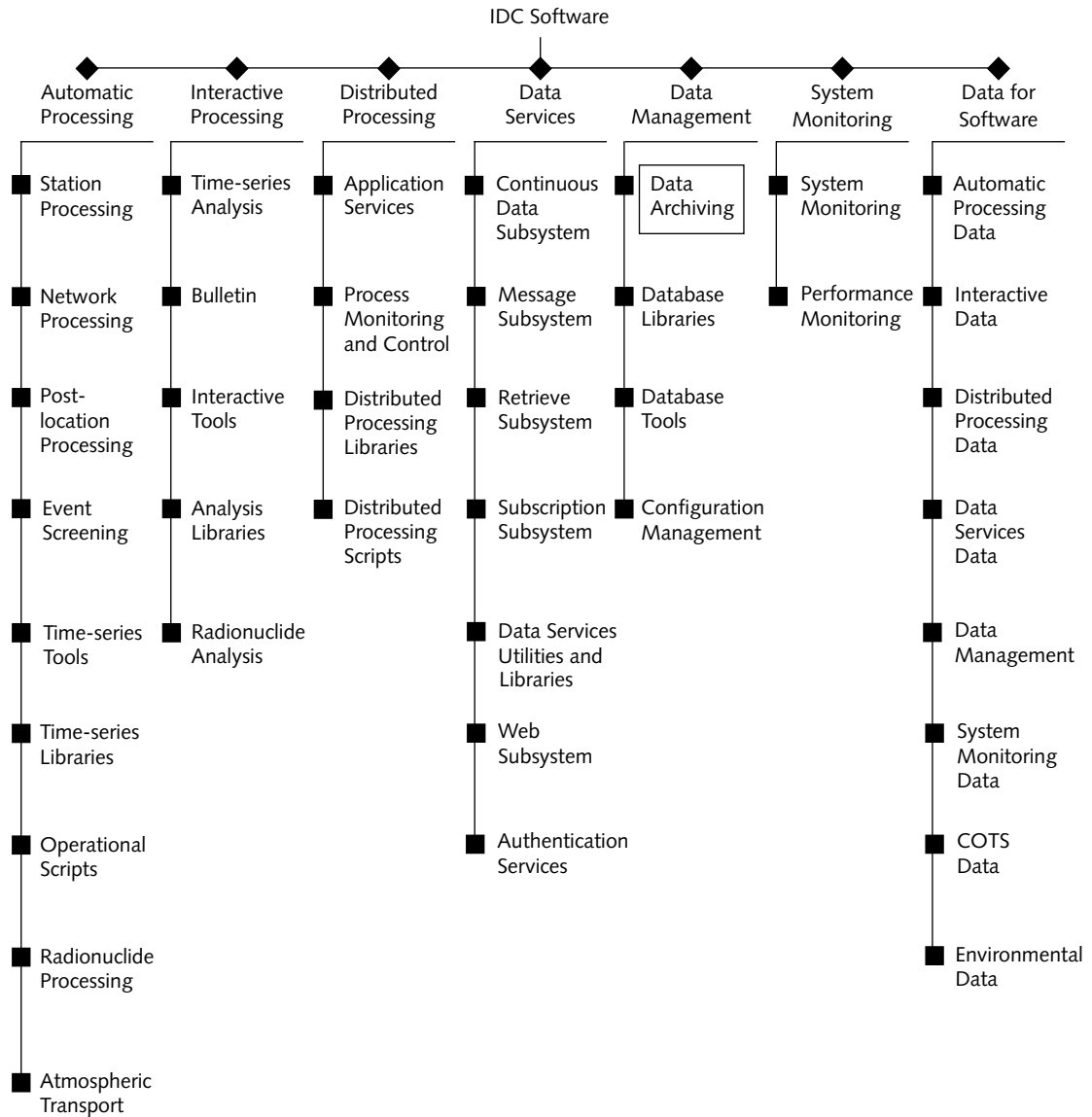


FIGURE 1. IDC SOFTWARE CONFIGURATION HIERARCHY

▼ Overview

Figure 2 shows the relationship of the Archiving Subsystem to the Continuous Data and Message Subsystems of the Data Services CSCI. The Archiving Subsystem moves data from the operations database and filesystem to the archive database and mass store filesystem.

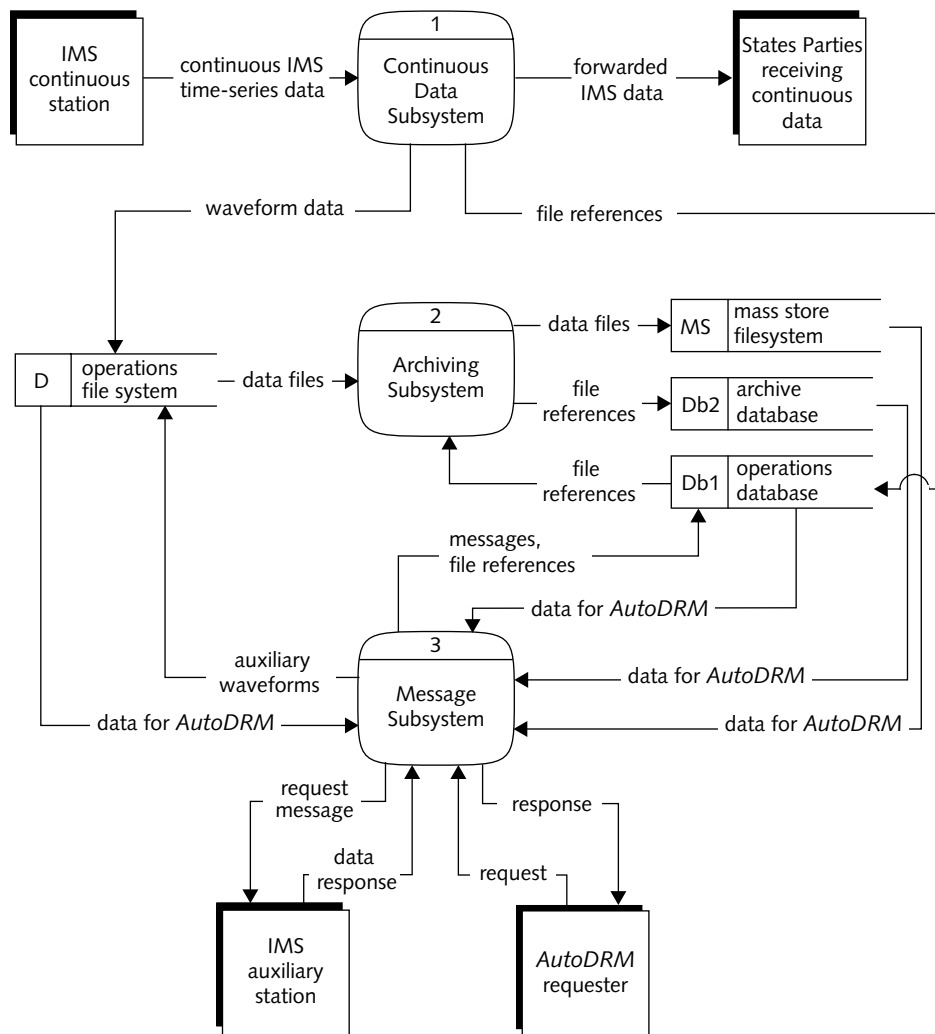


FIGURE 2. RELATIONSHIP OF ARCHIVING SUBSYSTEM TO CONTINUOUS DATA AND MESSAGE SUBSYSTEMS

FUNCTIONALITY

The Archiving Subsystem manages the transfer of data files from operational file-systems to the mass store filesystem. Data files (waveforms, messages, and fileproducts) are copied directly to the mass storage device. The associated database tables (**wfdisc**, **msgdisc**, and **fileproduct**) are transferred from the operations database to the archive database. Some attributes (that is, directory path, file name, and file offset) are modified to reflect the appropriate data location on the archive filesystem. *Archive* defines and queues the intervals to be archived, reads the database referencing the data, reads the data from the files, and writes to the archive hardware. *Archive* is monitored from *WorkFlow*. *MSwriter* is a light-weight server run on the archive hardware that manages the transfer of data and reports errors to the client software.

IDENTIFICATION

Components of the Archiving Subsystem are identified as follows:

- *Archive* (the archiving client)
- *MSwriter* (the archiving server)

STATUS OF DEVELOPMENT

The design of the Archiving Subsystem is complete. The initial implementation of the Archiving Subsystem addresses almost all requirements defined in Chapter 4: Requirements. Minor extensions to the initial implementation are under development to address some outstanding requirements.

BACKGROUND AND HISTORY

David Salzberg of SAIC developed the Archiving Subsystem in 1999.

The Archiving Subsystem was first installed in operations at the Prototype International Data Centre (PIDC) at the Center for Monitoring Research (CMR) in Arlington, Virginia, U.S.A. in September 1999. It has been used continuously at the PIDC since that time. The Archiving Subsystem was delivered to the International Data Centre of the Comprehensive Nuclear-Test-Ban Treaty Organization (CTBTO IDC) in Vienna, Austria in November 1999. It is anticipated that the IDC will install the Archiving Subsystem when the mass storage device has been fully commissioned.

OPERATING ENVIRONMENT

The following paragraphs describe the hardware and commercial-off-the-shelf (COTS) software required to operate the Archiving Subsystem.

Hardware

The Archiving Subsystem does not require significant system resources (32 MB RAM, disk space for log files, run-time parameters) and runs locally on the system hosting the data that are to be archived. With continuous waveform archiving, for example, *Archive* resides on the Disk Loop Host (DLHOST), which is a Sun E4000 Server with approximately 115 GB of disk. The storage requirements for the DLHOST are determined by the storage needs of the disk loop data files and are independent of the Archiving Subsystem. Similarly, the processing load is driven by signal processing of the disk loop data. *MSwriter* resides on the mass storage server. This server has disk space to operate the server as well as the storage associated with the mass storage device itself. *MSwriter* only requires a modest amount of disk space, whereas the storage requirements for the mass storage device are determined by the data accumulation rate. Figure 3 shows a representative hardware configuration.

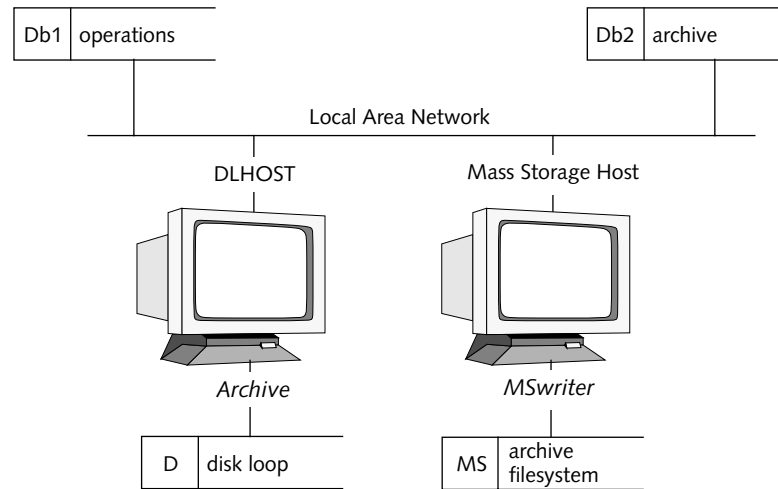


FIGURE 3. REPRESENTATIVE HARDWARE CONFIGURATION FOR ARCHIVING SUBSYSTEM

Commercial-Off-The-Shelf Software

The Archiving Subsystem software requires a relational database both for its internal processing and as a source and destination for migration of the associated database tables. The software was tested using ORACLE 8i. The software runs on a Solaris 7 operating system.

Chapter 2: Architectural Design

This chapter describes the architectural design of the Archiving Subsystem and includes the following topics:

- Conceptual Design
- Design Issues
- Functional Description
- Interface Design

Chapter 2: Architectural Design

CONCEPTUAL DESIGN

The purpose of the Archiving Subsystem is to facilitate the permanent storage and retrieval of all data arriving at the IDC. At the end of the archiving process the data (filesystem objects) reside on the mass store and the database references to the data reside in the archive database. While the process is conceptually simple, the Archiving Subsystem also must organize the data for efficient retrieval from the mass store, must verify that the data are archived correctly, and must include a mechanism for error recovery.

Figure 4 is a conceptual model of the Archiving Subsystem. The subsystem moves data from the operations disk storage filesystem to the archive storage medium. All database references to the data are also copied from the operations database to the archive database in a manner that ensures data accessibility at a later time. *WorkFlow* is used to monitor the status of archiving.

Because the archive hardware resides on a file server separate from the operational filesystem, a mechanism is provided to transmit data from the operational server to the archive server. The archive hardware is assumed to use a UNIX filesystem, where the data files can be referenced by directory, filename, and byte offset. A socket connection is used to isolate the archive hardware from the operational system.

The Archiving Subsystem may be configured to archive many different data types. Table 1 lists the supported data types and the nominal waiting period prior to archiving.

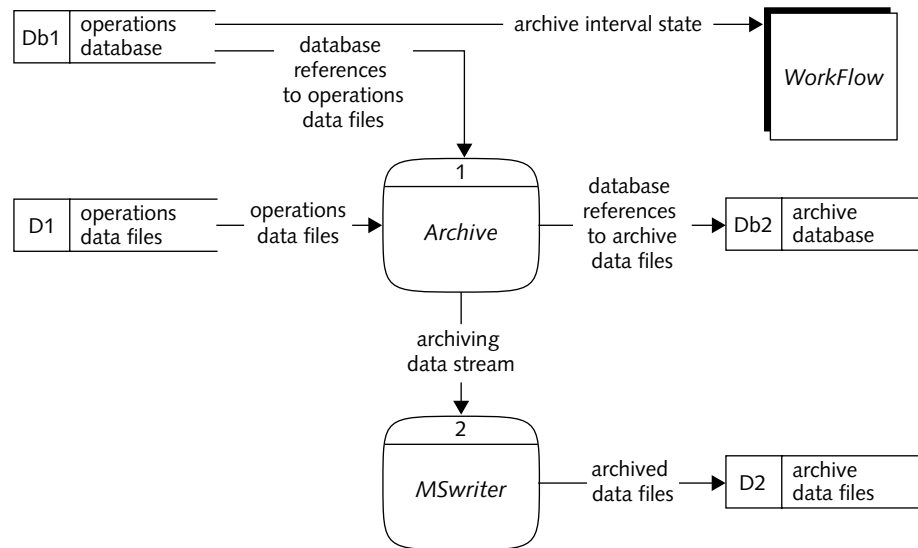


FIGURE 4. CONCEPTUAL MODEL OF ARCHIVING SUBSYSTEM

TABLE 1: SUPPORTED CONFIGURATIONS OF ARCHIVING SUBSYSTEM

Product	Nominal Waiting Period
Continuous Waveform Data (CD-1.0, CD-1.1)	archived after 3 days
Auxiliary waveforms	archived after 7days
Messages and Fileproducts	archived after 10 days

DESIGN ISSUES

The following design decisions pertain to the Archiving Subsystem.

Programming Language

Each software unit of the Archiving Subsystem is written in the C programming language unless otherwise noted in this document.

▼ Architectural Design

Global Libraries

The software of the Archiving Subsystem is linked to the following shared libraries: *liblogout*, *libpar*, *libinterp*, *libstdtime*, and *libgdi*.

The software is also linked to the following COTS libraries: *libsocket*, *libnsl*, *libm*, and *libdl*.

Database

The Archiving Subsystem uses the ORACLE database for managing the archiving process and for tracking the archiving status, as well as for reading and writing database records referencing the archived files.

Interprocess Communication (IPC)

The Archive Subsystem does not use the message services of the Distributed Application Control System (DACS). However, the Archiving Subsystem may be initiated from DACS.

Filesystem

The filesystem holds the run-time parameters (par files) of the Archiving Subsystem. In addition, both the pre- and post-archiving file objects are stored on the filesystem.

Design Model

The design of the Archiving Subsystem is primarily influenced by flexibility and reliability requirements. The flexibility is achieved by the subsystem's ability to manage the archiving of multiple data types (waveforms, messages, and so forth), and by the use of the **interval** table, which eliminates the need for specialized code to monitor the archiving process. All interfaces with other subsystems are handled by the database. Reliability is achieved by using a transactional processing mechanism in all processing steps; if any single step fails, the Archiving Subsystem attempts to roll back to the original state.

Database Schema Overview

The Archiving Subsystem maintains all key information in an ORACLE database. Three database tables, **arch_data_type**, **chan_groups**, and **lastid_arcdb**, are unique to the Archiving Subsystem and provide information for archive processing. The file-system objects that are processed by the Archiving Subsystem are referenced in the database. The database also stores the information, allowing the subsystem to determine what data have already been archived.

Table 2 shows the database tables used by the Archiving Subsystem along with a description of their use. The Name field identifies the database table. The Mode field is “R” if the Archiving Subsystem reads from the table and “W” if it writes to the table. The abbreviation following the mode indicates the database in which the table resides; “Ops” is the operations database, and “Arch” is the archive database.

TABLE 2: ARCHIVING SUBSYSTEM DATABASE TABLE USE

Name	Mode	Description
arch_data_type	R/W Ops	This table stores information representing each data type (or class of data) that is supported.
chan_groups	R/W Ops	This table stores station and channel pairs grouped by class and name, allowing the Archiving Subsystem to determine how the data should be grouped.
dlfile	R Ops	This table describes the files in the disk loops that are managed by the <i>DLMan</i> and <i>DLParse</i> applications.
fileproduct	R Ops/ W Arch	This table contains descriptions of files containing products.
fpdescription	R Ops	This table contains descriptions of product types used with file products.
interval	R/W Ops	This table defines the time intervals for processing.

▼ Architectural Design

TABLE 2: ARCHIVING SUBSYSTEM DATABASE TABLE USE (CONTINUED)

Name	Mode	Description
lastid	R/W Ops	This table contains counter values (last value used for keys) and is a reference table from which programs may retrieve the last sequential value of one of the numeric keys. Unique keys are required before a record can be inserted in numerous tables. lastid has exactly one row for each <i>keyname</i> . problastid and rms_lastid are views of the lastid table.
lastid_arcdb	R/W Arc	This table is unique to the Archiving Subsystem and resides on the archive database. It is used by the <i>Archive</i> application for obtaining unique <i>wfid</i> values.
msgdisc	R Ops/ W Arch	This table contains information pertinent to <i>AutoDRM</i> messages including the date and time that the message was sent or received, identification information, and where the message is stored.
wfaux	R Ops/ W Arch	This table includes the size (in bytes) of waveform files.
wfdisc	R Ops/ W Arch	This table contains waveform header and descriptive information as well as a pointer (or index) to waveforms stored on disk.

FUNCTIONAL DESCRIPTION

The primary function of the Archiving Subsystem is to facilitate permanent storage of data in a manner ensuring complete data archiving while allowing for data retrieval at a later time. The data types currently configured for archiving are listed in Table 3. Other data types, such as CD-1.1 Frame Store, Radionuclide Data, and Threshold Monitoring Results, can be configured using the **fileproduct** interface.

TABLE 3: SUPPORTED DATA TYPES AND DATABASE TABLES

Datatype	Database Table
Waveforms	wfdisc, wfaux
IMS Messages	msgdisc
Station and Channel Capability Reports	fileproduct

The archiving process is straightforward; raw data are copied from the operational filesystem to the archive filesystem (usually mounted on some type of mass storage device). The corresponding database records are read from the operations database, updated to reflect the new file location, and written to the archive database. The Archiving Subsystem organizes the data during the archiving process to allow easy retrieval. In addition, because failure to archive results in permanent loss of data, the Archiving Subsystem monitors the status of archiving.

A conceptual diagram of the Archiving Subsystem is shown in Figure 5. This view is based on processes, which are not software units. The processes relate to the software units in a manner discussed in "Internal Data Flow" on page 23. The first process (*Create Intervals*) generates new intervals to process by setting the **interval.state** attribute to **NEW**. The intervals are then queued for processing in the *Queue Intervals* process by updating the **interval.state** to **QUEUED**. The Archiving Subsystem then loops over each interval in the *Run Interval* process. If the interval is successfully archived, the **interval.state** is set to **DONE**, otherwise the **interval.state** is set to **FAILED**. The following sections describe these processes in more detail.

Create Intervals

The *Create Intervals* process generates new **interval** rows for archiving. First, database queries are obtained for a specific type of data (*datatype*) from a run-time parameter or a parameter file. *Create Intervals* then executes this query and stores the resulting intervals in internal data structures (using a new interval primary key

▼ Architectural Design

[*intvlid*]). The internal interval data structures are then inserted into the **interval** table in the operations database. The basis for forming the intervals and the initial value of the *state* attribute are defined in parameter files.

Queue Intervals

The *Queue Intervals* process determines the processing intervals that are ready to be archived and inserts them into a data structure. The first action is accomplished by executing a database query, which updates the *state* attribute of the **interval** table from *state* NEW to *state* QUEUED. The database query, which represents the queuing rules, is obtained from the parameter file. The queued intervals are then inserted into an internal interval data structure.

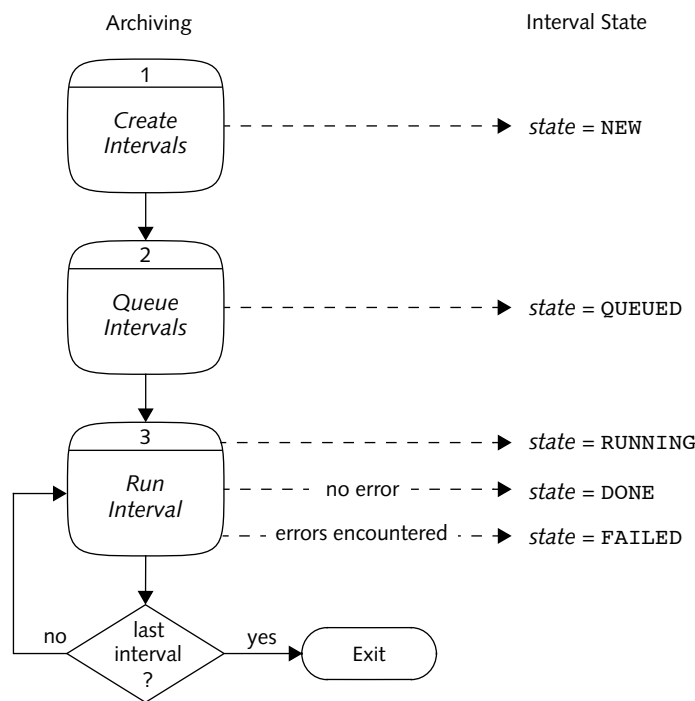


FIGURE 5. ARCHIVING SUBSYSTEM INTERVAL PROCESSING AND STATUS

Run Interval

Run Interval processes each of the queued intervals sequentially as shown in Figure 6. The first action in *Run Interval* is to update the **interval.state** attribute to **RUNNING**. *Run Interval* then reads the operations database references into data structures. The database query for reading the data structures is specific to the particular data type and is obtained from the parameter file. Next, *Run Interval* generates the data structures that will reference the data after they are in the archive database. With most data types, this involves duplicating the operational data structures and modifying the directory name, the filename, and the byte offset; however, with waveform data types the software can be configured to merge adjacent records. The details of merging adjacent records are discussed in “Merge-Data” on page 37.

After the archived data structures are constructed, the data are read from the operations data file referenced in the operational data structures. The data are then transmitted to the application *MSwriter* through a socket connection. *MSwriter* writes the data to a mass storage device and reports its final status to the *Run Interval* process. *Run Interval* then adds the data structure that represents the archived data to the archive database. Finally, *Run Interval* updates the **interval.state** attribute to **DONE**. If any errors are encountered the process aborts and the **interval.state** attribute is set to **FAILED**.

Monitoring

Because the Archiving Subsystem is managed through the *state* attribute in the **interval** table, the archiving process may be monitored through *Workflow*. This eliminates the need for special code to monitor the Archiving Subsystem’s progress.

▼ Architectural Design

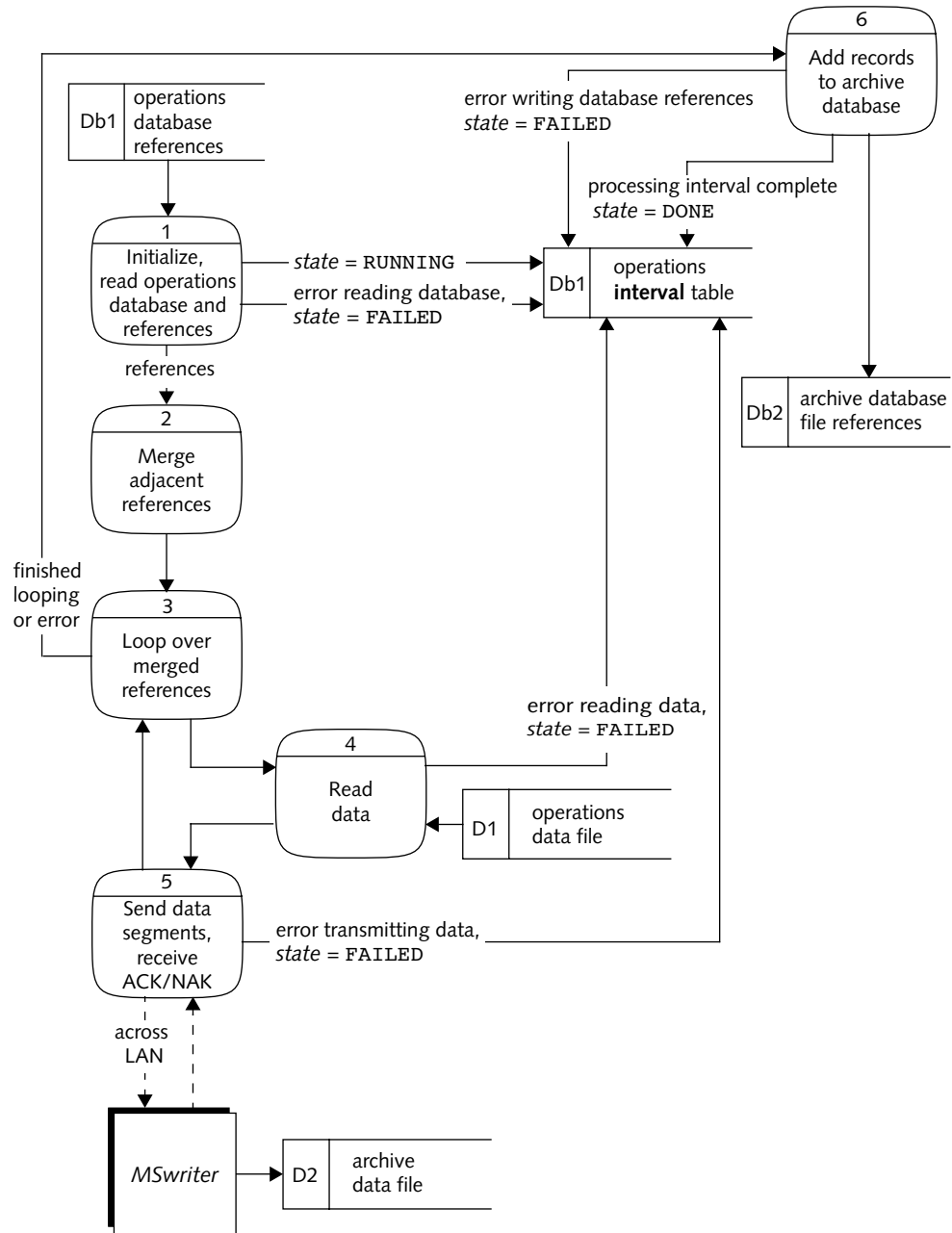


FIGURE 6. PROCESSING FLOW FOR RUN INTERVAL

INTERFACE DESIGN

This section describes the Archiving Subsystem's interfaces with other IDC systems, external users, and operators.

Interface with Other IDC Systems

The Archiving Subsystem interfaces with other IDC systems for storing data on the mass store, monitoring the archiving process, and obtaining the data to be archived. The mass store appears as a standard UNIX filesystem. The interface with the monitoring systems is via the *state* attribute in the *interval* table. Each step of archiving processing updates the *state* attribute of the interval table. *WorkFlow* (in the DACS) monitors and presents, through a graphical user interface (GUI), the interval status.

The Archiving Subsystem obtains the data to be archived by reading particular database tables and the corresponding filesystem objects. The interface is fairly straightforward: the database includes the location (directory name, filename, byte offset, and the number of bytes) of the data. In addition, the database includes the storage format of the data. Based on that information, the Archiving Subsystem is able to read the filesystem object.

Three database tables are supported for the purpose of identifying the information to be archived: **wfdisc**, **msgdisc**, and **fileproduct**. These tables support various data types, as listed in Table 3 on page 15. The **wfdisc** table is used to describe the waveform data files and is the primary interface for continuous waveform data and auxiliary waveform data. The **msgdisc** table is used to describe the IMS messages used by the message subsystem. With both waveform and message archiving the Archiving Subsystem assumes that the **wfdisc** and **msgdisc** table attributes correctly reference the data being archived.

The **fileproduct** table is used to handle any other type of data that must be archived. Any filesystem object can be described as a fileproduct. Currently, six types of fileproducts have been identified: Station and Channel Status reports, radionuclide data, radionuclide reports, Threshold Monitoring results, and CD-1.1

▼ Architectural Design

Frame Stores. Additional fileproducts can be created as needed. The procedure to define a fileproduct, with an example, is presented in “Appendix: Defining Fileproducts” on page A1.

Interface with Internal Users

The Archiving Subsystem has no interfaces with internal data center users. However, internal users with accounts that allow direct access to the archive database and mass storage device may require access to the data archived by the Archiving Subsystem. Access to archived data is through the data references (**wfdisc**, **msgdisc**, and **fileproduct**).

Interface with External Users

The Archiving Subsystem has no direct interfaces with external users. However, external users who require access to the archived data may interface via the *AutoDRM* process of the Message Subsystem [IDC7.4.2]. External users request data via the Message Subsystem using the IMS 1.0 Formats and Protocols [IDC3.4.1Rev2]. *AutoDRM* first checks the operations database for the data. If data are not present, *AutoDRM* then queries the archive database. If the archive database contains rows that satisfy the request, *AutoDRM* accesses the data files, which reside on the mass storage device, through *libwfm*.

Interface with Operators

The Archiving Subsystem can be monitored and maintained through the *WorkFlow* process of the Distributed Application Control System (DACS) [IDC7.3.1] because the **interval** table is used to store the state of the processing. IDC Operators monitor the software for old intervals that are not in state **DONE** or for any **FAILED** intervals. Additional information is available in log files of the *Archive* and *MSwriter* processes.

Chapter 3: Detailed Design

This chapter describes the detailed design of the Archiving Subsystem and includes the following topics:

- Data Flow Model
- Processing Units
- Database Description

Chapter 3: Detailed Design

DATA FLOW MODEL

The data flow of the Archiving Subsystem can be viewed from several perspectives: the data flow in the context of external components, the data flow in the context of internal processing, and the data flow in the context of internal data exchange. The role of the Archiving Subsystem is to migrate data files to the mass store and create database records in the archive database that point to the archived files. The Archiving Subsystem fulfills its role by a defined sequence of steps to minimize the risk of data loss.

External Data Flow

The external data flow is shown in Figure 7. The data are imported or created using other subsystems (for example, Continuous Data Subsystem, Message Subsystem, or System Monitoring), resulting in files on the filesystem and corresponding database records pointing to the files. The Archiving Subsystem reads the operations database records and corresponding data files and copies the data to the mass storage hardware. After receiving acknowledgment of successful processing, the Archiving Subsystem inserts new database records in the archive database that reference the archived data.

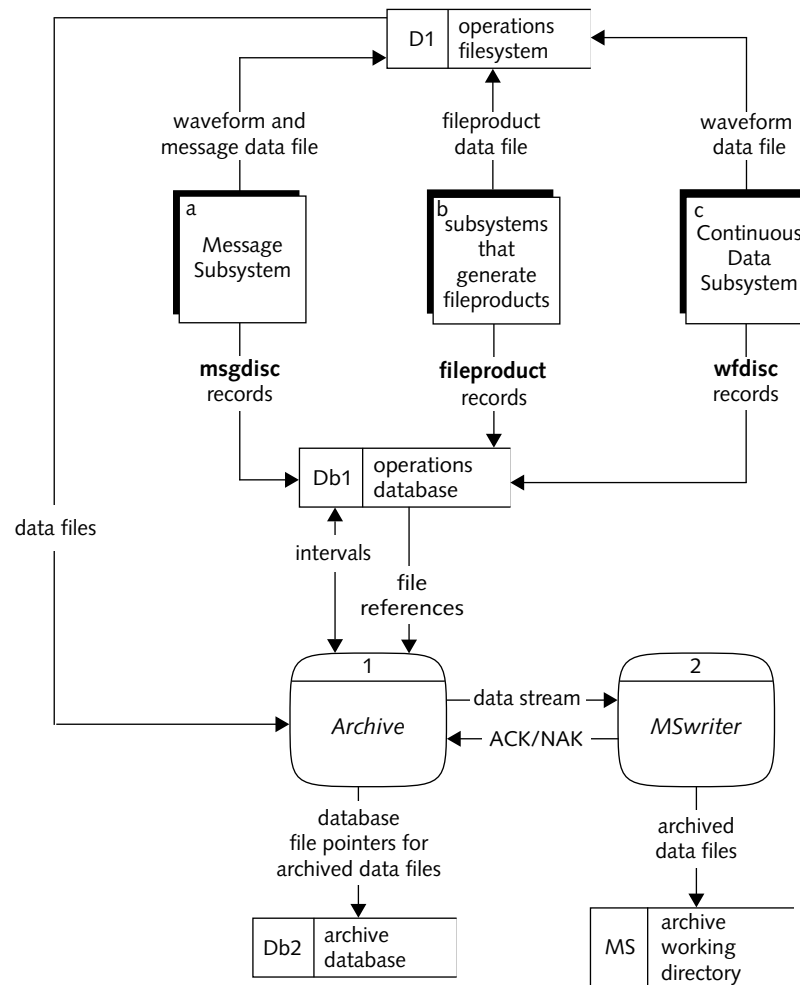


FIGURE 7. DATA FLOW OF ARCHIVING SUBSYSTEM

Internal Data Flow

The internal data flow is shown in Figure 8, including both the functional data flow and the data flow in the context of software units. Processes in the Figure are indicated with two-word titles and software units have single-word names. Several processes may be performed by a single software unit (for example, the processes

▼ Detailed Design

Create Intervals, *Queue Intervals*, and *Read Intervals* are performed by the software unit *ManageInterval*) or a single process may be performed by several software units (for example, the process *Run Interval* is performed by the software units *GetData*, *MergeData*, and *ReadWriteData*).

The first step in the internal data flow is the initialization of the archiving process. Initialization involves reading the run-time parameters, opening the database connections, and reading the parametric data stored in the database table **arch_data_type**. One required run-time parameter is *datatype*. Some of the parametric values are based on the value of the *datatype* parameter. In particular, most of the database queries are datatype-specific; for these, the parameter name includes the datatype (for example, **PRIARC-interval_create** gives the parameter for creating intervals for *datatype* PRIARC).

The next step is to identify what data must be archived. This step is handled by the software unit *ManageInterval* in the process *Create Intervals*. The basis for this step is to execute the datatype-specific parameterized interval creation query. This query identifies new intervals by comparing the time range of the referencing database records to existing intervals. If the interval does not yet exist, the process *Create Intervals* generates a new database row in the **interval** table.

After the intervals are identified, the next step is to determine when data are ready to be archived. Because the system uses the **interval** table to identify, queue, and track the archiving process, the data are queued for archive processing by updating the *state* attribute in the **interval** table. Thus, the primary role of the process *Queue Intervals* (in the software unit *ManageInterval*) is to execute database queries that update the *state* attribute of the **interval** table to *state QUEUED* based on the database query that is specified in datatype-specific parameters. The typical queuing compares the interval time to the current time; if the difference is greater than a value specified in the database query, then the interval is queued.

The software unit *ManageInterval* then reads all the queued intervals for the datatype (or **interval.class**) being archived. The Archiving Subsystem then processes the intervals in time order.

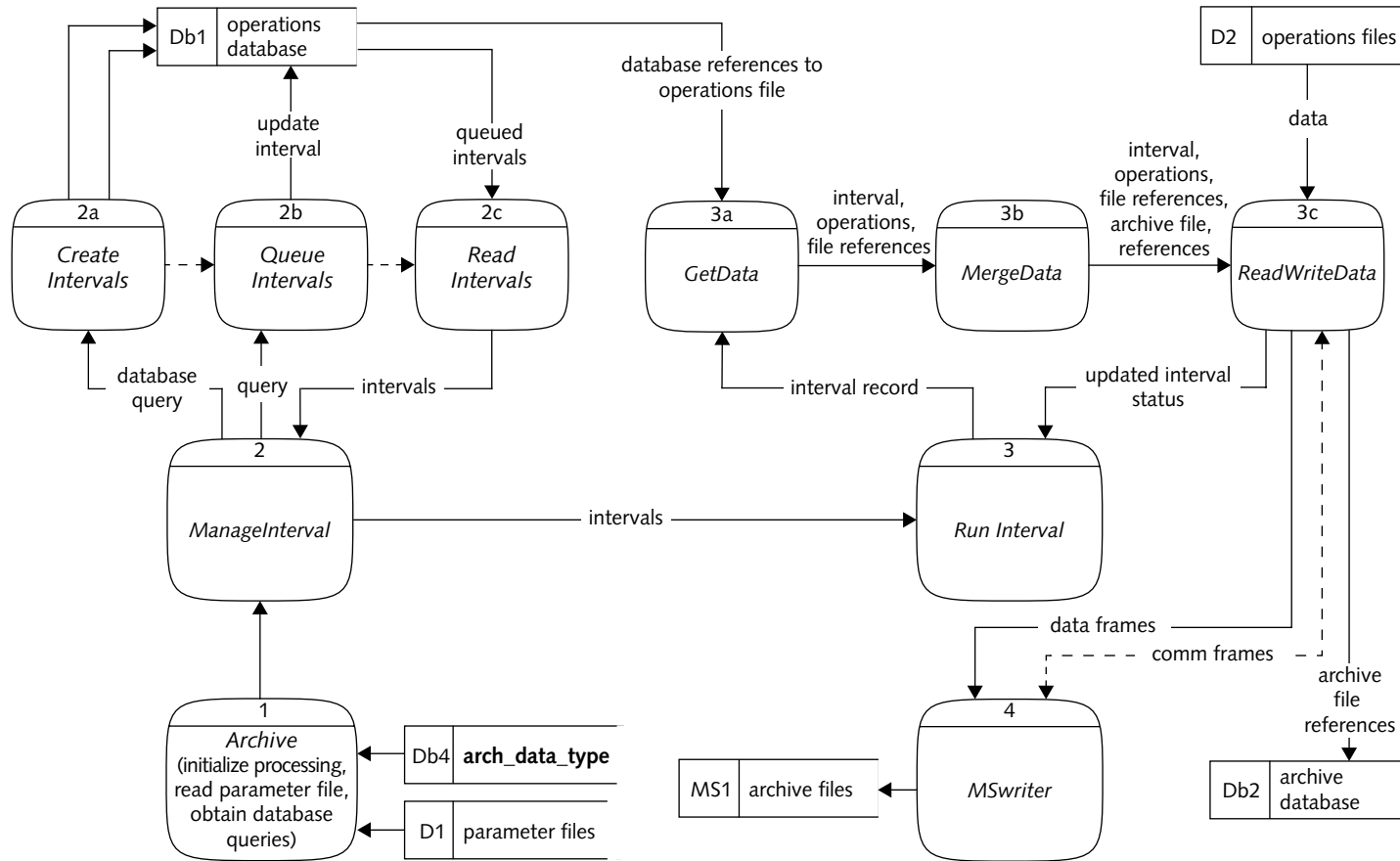


FIGURE 8. DATA FLOW OF ARCHIVE PROCESSING

▼ Detailed Design

The Archiving Subsystem processes each interval sequentially, executing the process *Run Interval*. The data flow in this functional unit can best be understood by examining the data flow from the software component perspective. First, *Archive* updates the *interval.state* to *RUNNING*. Next, *Archive* calls *GetData*, which reads in the database records that reference the data that are being archived. *Archive* then invokes the software component *MergeData*. *MergeData* merges adjacent database records and generates new data structures that reference the archived data. The new archive data structures are based on the operational data structures. Fileproducts or messages are initialized by duplicating the data structure. For waveform datatypes, the software can be configured to merge adjacent waveform data by setting the attribute *merge_data* to *y* in the *arch_data_type* table. In this mode, if two *wfdisc* data structures for the same channel are adjacent (the first sample referenced in one data structure is one sample later than the last sample in the previous structure), then the two structures are combined into one.

Some of the data structure attributes (*dir*, *dfile*, and *foff*) are adjusted for the new location of the data after they are archived. The adjustments occur as the first step in *ReadWriteData*. *ReadWriteData* then reads the data from the data file, generates a checksum, and sends the data to *MSwriter* using the archive protocol. *MSwriter* verifies the checksum, creates new directories and files that are needed, and writes the data to a local filesystem (usually a mass storage device). When this operation is completed successfully for all data in an *interval*, *ReadWriteData* inserts the new data structures that reference the archived data into the archive database. If an error occurs in the file transfer or in the database write, the process is aborted and *Archive* updates the *interval.state* to *FAILED*; otherwise, it is updated to *DONE*. If an error is encountered, no effort is made to remove the data file, which may have been created on a write-only archive filesystem.

Flow of Internal Data Exchange (Archive Protocol)

The data flow between the archiving client (*Archive*) and the archiving server (*MSwriter*) follows the archive protocol, which provides a robust method for transferring data between the client and server. The archive protocol relies on the exchange of data structures, which are referred to as “frames.” Each transmission

begins with a communication frame using the format defined in Table 4. The communication frame indicates the type of the next frame that will be sent from that source (either client or server) in the *xtype* field (see Table 5). A communication frame with *xtype* = C acknowledges communications and indicates status (see Table 6). A communication frame with *xtype* = A, F, P, or D is immediately followed by another frame. Table 7 lists the structure of an archive timestamp frame, which follows a communication frame with *xtype* = A. The timestamp frame is used to verify that *Archive* and *MSwriter* are synchronized. If the timestamp from *Archive* differs by more than 100 seconds from the time it is received by *MSwriter*, then the connection is dropped (this requires that the system clocks on the *Archive* and *MSwriter* hosts agree). File headers and data segment headers (which follow a communication frame with *xtype* = F and P, respectively) use a common structure, as shown in Table 8. When *xtype* = D, a data frame follows. A data frame contains raw bytes; the size and the checksum of the data frame are defined in the communication frame.

TABLE 4: STRUCTURE OF COMMUNICATIONS FRAME

Name	Storage Type	Description
<i>fsn</i>	long	frame sequential number
<i>xsize</i>	long	frame size
<i>xchksum</i>	long	frame checksum
<i>status</i>	int	status (when <i>xtype</i> = C)
<i>xtype</i>	char	type of the following data frame

▼ Detailed Design

TABLE 5: DATA FRAME TYPES FOR COMMUNICATIONS FRAME

xtype	Frame Type
A	timestamp frame
C	communications frame
D	data frame
F	file header frame
P	data segment header frame

TABLE 6: VALID STATUS VALUES FOR COMMUNICATION FRAMES¹

Values (from an enumerated list)	Type	Description
ARCH_PROC_OK	info	system is behaving normally
Arch_retransmission	info	<i>Archive</i> is informing <i>MSwriter</i> that it will retransmit a segment
Send_cksum_fail	warn	checksum operation failed
Dir_make_fail	error	server is unable to make directory
File_open_fail	error	server is unable to open file
File_write_fail	error	server is unable to write the file
Server_io_fail	fatal	an unknown error condition was encountered

1. Only applicable for *xtype* = C.

TABLE 7: STRUCTURE OF ARCHIVING SUBSYSTEM TIMESTAMP FRAME

Name	Storage Type	Description
<i>etime</i>	epoch_t	time of message for receipt valid
<i>starttime</i>	long	this field is not used
<i>endtime</i>	long	this field is not used

TABLE 8: STRUCTURE OF FILE AND DATA SEGMENT HEADER FRAMES

Name	Storage Type	Description
<i>dir</i>	char[65]	directory of archived data file
<i>dfile</i>	char[33]	name of archived file
<i>foff</i>	long	file offset (in bytes)
<i>size</i>	long	size of file (in bytes)

The data flow relating to the Archiving Subsystem protocol is shown in Figure 9. First, the *Archive* component *ReadWriteData* opens the socket connection. *MSwriter* is defined as an *inetd* daemon, and therefore opening the socket connection invokes an instance of *MSwriter* on the mass storage server. *Archive* then transmits a communication frame with *xtype* = A followed by a timestamp frame. *MSwriter* acknowledges receipt by sending back a communication frame with *xtype* = C. If the acknowledgment indicates successful completion of the operation, *Archive* transmits a communication frame with *xtype* = F followed by a file header frame. *MSwriter* makes a new directory (if necessary) and opens the file for writing. If any of these actions fail, *MSwriter* reports the appropriate error condition to *Archive* in a communication frame with *xtype* = C and *status* from the enumerated list shown in Table 6; otherwise, *MSwriter* reports success in the communication frame (*status* = ARCH_PROC_OK). *Archive* then transmits a communication frame with *xtype* = P followed by a segment header frame, which tells *MSwriter* how much data to expect. This frame is again acknowledged by *MSwriter* using a communication frame with *xtype* = C. Next, *Archive* transmits a communication frame with *xtype* = D, which indicates that data are coming and contains the checksum of the data. *Archive* then sends the data stream to *MSwriter*. *MSwriter* validates the data by performing a checksum and comparing it to the *xchksum* value sent in the communication frame. If there is a discrepancy, *MSwriter* sends a communication frame indicating that checksum failed, and *Archive* retransmits the data. If the checksum continues to fail after a parameter-settable number of retries (the default is 6), *Archive* generates an error condition and aborts processing; otherwise,

▼ Detailed Design

MSwriter reports success, and *Archive* transmits the next segment. Each segment is a maximum of 64 KB (65,536 bytes); that is, a data file is transmitted in 64 KB chunks. This value is defined in a header file.

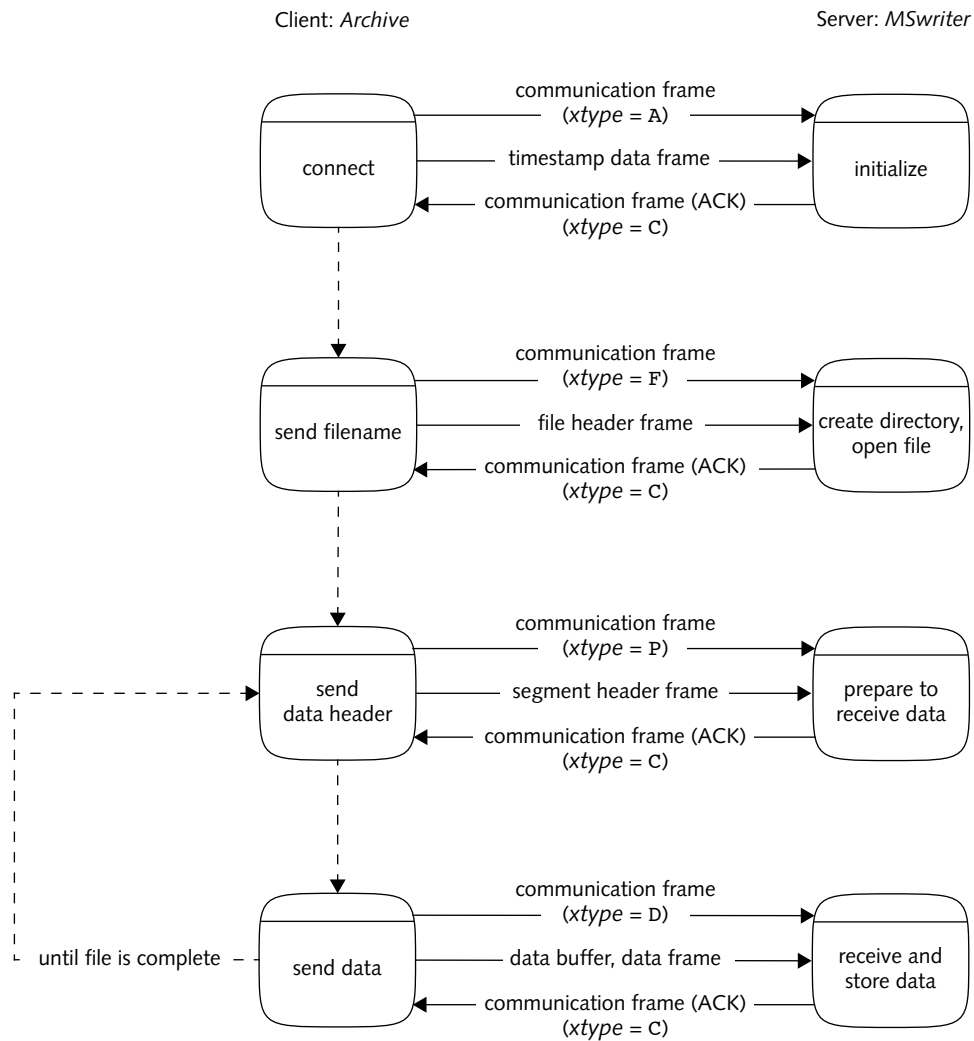


FIGURE 9. DATA FLOW OF PROTOCOL EXCHANGE BETWEEN ARCHIVE AND MSWRITER

PROCESSING UNITS

The Archiving Subsystem software consists of the following processing units:

- *Archive*
- *ManageInterval*
- *GetData*
- *MergeData*
- *ReadWriteData*
- *MSwriter*

ManageInterval, *GetData*, *MergeData*, and *ReadWriteData* are functions called by *Archive*.

The following paragraphs describe the design of these units. The logic of the software and any applicable procedural commands are also provided.

Archive

Archive initiates the archiving process. It first parses the run-time parameters, which include datatype-specific parameters. *Archive* then calls the four functions: *ManageInterval*, *GetData*, *MergeData*, and *ReadWriteData*, which are described in the sections that follow. *Archive* is the parent of the archiving process. It calls functions that manage intervals, read from the operational filesystem (and database), transmit data to *MSwriter* for archiving, and write the archive database records referencing the archived data.

Input/Processing/Output

Archive reads the **arch_data_type** database table for the *datatype* and the **wfdisc**, **msgdisc**, and **fileproduct** database tables, which reference the raw data. *Archive* also reads the data on the filesystem, intervals from the **interval** table, and the system time. Finally, *Archive* reads the run-time parameters, which include datatype-specific parameters such as the parameters that contain the queries that control the database access.

▼ Detailed Design

The outputs of the archiving process are new intervals, log files, and database records that reference the archived data. In addition, data are transmitted over a socket connection to *MSwriter* for permanent storage on the mass storage device.

The first step in the *Archive* process is to parse the run-time parameters. Next, *Archive* reads the **arch_data_type** row for the parameter-specified *datatype*. *Archive* then creates and queues intervals via the function *ManageInterval*. Functions called by *Archive* also read the operations database records (*GetData*) into internal data structures and generate the data structures that reference the archived data (*MergeData*). The archive data structures are generated by copying the operations database references, merging adjacent data structures (for time-series data), and updating the attributes *dir*, *dfile*, and *foff* to reflect the location of the archived data. Finally, *Archive* reads and transmits the data to *MSwriter* via the function *ReadWriteData*, and inserts the archive data structures into the appropriate database table in the archive database.

Control

Archive can be initiated from the command line. However, the command to initiate archiving is typically invoked from *cron*. Run-time parameters (most of which are stored in parameter files) are used to specify the datatype being archived and the archiving rules. Some of the parameter names are datatype-specific, because the value of the *datatype* parameter is used to determine the parameter name. For example, the parameters contain SQL (Structured Query Language) statements that define the rules for both interval creation and queueing as well as reading the database records that reference the data. *Archive* also relies on feedback from *MSwriter* to ensure that data were properly archived.

Interfaces

Archive interfaces with the archive filesystem through *MSwriter* by socket connection using the archive protocol. *Archive* interfaces with external systems (for example, Continuous Data Subsystem, or Message Subsystem) through database records (for example, **msgdisc**, **wfdisc**) and associated data files.

Error States

Error handling is fundamental to the architecture of *Archive*. Because archiving data is one of the core missions of the IDC software, it is critical that *Archive* accurately report its processing status. The approach used to accomplish this is to make *Archive* transactional; *Archive* first determines what data are ready to be archived and establishes intervals. *Archive* then reads the reference database records. Next, *Archive* transmits the data via the archive protocol to *MSwriter*. Only after *MSwriter* reports to *Archive* that processing was successful (and all of the data files in an interval were archived) does *Archive* add the new reference rows to the archive database. The *interval.state* is updated to `DONE` when all other steps have completed. If any error occurs the *interval.state* is set to `FAILED` and the process is aborted. The log file indicates where the problem occurred.

There are several places where errors can occur during *Archive* processing. First are configuration errors, which include incorrect database names or passwords and errors in the datatype-specific queries. Configuration errors are recorded in log files. These errors typically occur prior to processing intervals. While processing intervals there can be failures to read from the database, to read the data file, to transmit to the archive hardware, to write the file or create the directory, or to write the reference database rows.

The following list describes the common errors, causes, and defensive programming:

- Failure in data transmission to the *MSwriter*. These failures can result from hardware problems or local area network glitches. In such cases, the connection between *Archive* and *MSwriter* times out. *Archive* captures the error state and logs the problem. *Archive* aborts processing and updates *interval.state* to `FAILED`. The only fix is to repair the hardware or networking problem and re-archive the data.
- Error writing to the mass storage device. These errors indicate *MSwriter* is unable to create a file or make a directory. *Archive* writes the error to a log file, aborts processing, and updates *interval.state* to `FAILED`. The

▼ Detailed Design

problem is most likely the result of improper permission settings on the mass storage device. To fix the problem validate and/or properly set the file access permissions, and archive the interval again.

- Failure to read from an operational file. This failure occurs when the system is unable to read from the operational file or an error is encountered when reading from the file. When this occurs, *Archive* logs the error and aborts processing. The problem has two probable causes: file permissions are not set properly or the expected file size recorded in the database is different than the actual file size of the data file. The fix to the first problem is to verify and set the permissions properly. The fix to the second problem is to clean up the database records, such that either the file size matches the actual size or the database records are purged. This problem has only been observed when the Message Subsystem attempts to write to a full filesystem partition. In one particular example, an error in the Message Subsystem resulted in the database records recording a 5-byte file size, whereas the filesystem had zero-length files.
- Error writing to the archive database. When this error occurs, *Archive* reports that it is unable to insert rows into the archive database. It then aborts processing and (if able to) updates **interval.state** to **FAILED**. This error condition results from the archive database being full, missing archive database tables, or the archive database instance being unavailable. The first and third problem can occur at any time. The second problem might occur immediately after installation. The repair for all three of these problems is to add space to the archive database, create a table, or restart the database instance, then rerun the interval.

ManageInterval

ManageInterval is a function called by *Archive*. *ManageInterval* handles the interval creation and queuing by executing database queries; the queries are parsed from run-time parameters (presumably stored in a parameter file). *ManageInterval* also sets the **interval.intvlid** attribute based on the **lastid.keyvalue**, where *keyname* = *intvlid*.

Input/Processing/Output

ManageInterval requires the following inputs: datatype-specific database queries that are obtained from the run-time parameters, system time, database tables that are specified in the parameterized database queries (which may join with additional tables, such as **chan_groups**), intervals, and the **lastid** table.

The processing steps include executing the database queries to create the new intervals by inserting the query results into an interval data structure. The primary key for intervals, *intvlid*, is assigned based on the *keyvalue* from the **lastid** table. The new intervals are then inserted into the **interval** table in the database. *ManageInterval* executes a query (from a run-time parameter) that updates **interval.state** to **QUEUED**. The final processing step in *ManageInterval* is to retrieve the database records for all of the **QUEUED** intervals for the particular *datatype* into a data structure. The resulting data structure and the number of queued intervals are the final output and are returned to *Archive* as arguments in the function call. In addition, *ManageInterval* returns the processing status **Arch_ok** if processing was successful. If a database error condition was encountered, then *ManageInterval* returns **Arch_opsdb failure**.

The outputs of *ManageInterval* are new intervals, which are inserted into the **interval** database table. In addition, queued intervals are returned to *Archive* as arguments in the *ManageInterval* function call.

Control

ManageInterval is initiated as a function call by *Archive*. The activity in *ManageInterval* is controlled by the values passed in the function call and the values in the associated database tables.

Interfaces

ManageInterval has two primary interfaces. It interfaces with the database and with *Archive* when *ManageInterval* is invoked. All of the database interactions are handled through the generic database interface (GDI). The interface with *Archive* is through the function call.

Error States

All of the potential errors encountered in *ManageInterval* involve failures in interacting with the database, either creating intervals, updating intervals, reading intervals, or obtaining the *lastid* value. The problems are either the result of a system failure (for example, database server failure), errors in the parameterized queries, or missing database tables. The symptom is the same; the query fails. With a failed query *Archive* logs the failure (including logging the query) and aborts processing.

GetData

GetData is a function called by *Archive*. *GetData* obtains the database records that point to the data to be archived.

Input/Processing/Output

GetData's input values are the *datatype*, the data structure containing the **interval** being processed, and the datatype-specific database query to get the data. *GetData* also obtains the database records referencing the data that are to be archived. The queries executed by *GetData* may join with other database tables (such as the **chan_groups** table) when archiving waveform data.

GetData incorporates the **interval.time** and **interval.endtime** into the query for reading the database record that references the data to be archived. *GetData* then selects the referencing database records and inserts them into datatype-specific data structures. *GetData* branches on the database table name for the database references (**wfdisc**, **fileproduct**, or **msgdisc**).

The outputs of *GetData* are the database records referencing the data. The reference database records are returned as elements in the function call. *GetData* also returns an error status through the function return.

Control

GetData is initiated as a function call by *Archive*. The activity in *GetData* is controlled by the values passed in the function call.

Interfaces

GetData has two primary interfaces. It interfaces with the database, and it also interfaces with *Archive* when *GetData* is invoked. All of the database interactions are handled through the GDI. The interface with *Archive* is through the function call.

Error States

The only error condition for *GetData* is a failure to read the database records from the operations database. In this case, it returns an error code to *Archive*. *Archive* logs that it is unable to read the database records, aborts processing, and updates *interval.state* to **FAILED**. The error is typically caused by a configuration error resulting from an incorrect database query in the parameter file.

MergeData

MergeData is a function called by *Archive*. The role of *MergeData* is threefold: (1) it duplicates the data structures referencing the operational data files, (2) it combines data structures that represent adjacent waveform segments into single segments, and (3) it maps the data structures referencing the operational data to the data structures that reference the data to be archived. This third step is particularly important when waveform segments are merged.

Input/Processing/Output

The inputs of *MergeData* are the data structures referencing the data to be archived, the number of reference structures, and the values in the database table **arch_data_type** (stored in a data structure). The first two types of input are arguments in the function call; the third is passed as a global variable.

▼ Detailed Design

The processing involved in *MergeData* is based on the type of data being archived. *MergeData* first duplicates the data structure containing the references to the data. This structure is datatype-dependent because different datatypes have different data structure sizes. For **wfdisc** records with the attribute *merge_adjacent* set to **Y** in the **arch_data_type** table, *MergeData* combines adjacent data structures of type **wfdisc** into a single data structure by modifying the elements *nsamp* (number of samples) and *endtime* (end time of the waveform segment). The modified data structure then represents a larger data segment, which is the union of the pre-merged segments. The algorithm used to merge adjacent waveform segments compares the *endtime* of one **wfdisc** data structure to the *time* of the next data structure record. If the time difference is within 0.5 multiplied by the inverse of sample rate of the expected time of the next sample, then *MergeData* assumes that the data were actually adjacent and combines the two data structures. Finally, *MergeData* sets up a hash structure that maps each operational data structure to an archive data structure. Multiple operations database records can map to one archive record.

The outputs from *MergeData* are the data structure referencing the data after archived, the number of these structures, the hash structure, and the number of elements in the hash structure.

Control

MergeData is initiated as a function call by *Archive*. The activity in *MergeData* is controlled by the values passed in the function call. In addition, *MergeData* branches on the database table name (or data structure type) and on a flag indicating whether adjacent records should be merged.

Interfaces

The only interface in *MergeData* is the interface with *Archive* through the function call.

ReadWriteData

ReadWriteData is a function called by *Archive*. The role of *ReadWriteData* is to send the data to the archive hardware and add the database records referencing the archived data to the archive database. *ReadWriteData* creates standard (datatype-independent) data structures that encapsulate the data files that are to be archived, the file offsets, and the file size. The standard structure excludes the datatype-specific information such as *sta* and *chan* for **wfdisc**. *ReadWriteData* also updates the attributes *dir*, *dfile*, and *foff* (or multiple *foff*'s for **msgdisc** rows) in the data structures that reference the archived data. These attributes are updated because the directory and filename are different after they are archived; in addition, because many operational files are combined into a single archived file, the file offsets need to be updated to reflect the new location within the data file. After the file reference data structures are set properly, *ReadWriteData* transmits the data to *MSwriter*. Upon completion of the data transmission (and only if successful) *ReadWriteData* inserts the data structures containing the reference database records into the archive database.

Input/Processing/Output

ReadWriteData has several inputs. A data structure containing the **arch_data_type** record is received as a global variable. The function call arguments for *ReadWriteData* include data structures referencing the data to be archived, the corresponding number of data structures, the data structures referencing the archived data once archived, the corresponding number of archive data structures, the hash table elements, the number of hash table elements, and a data structure containing the **interval** row. *ReadWriteData* also receives, through the socket connection, the status of the data transmission to *MSwriter*. The final inputs for *ReadWriteData* are the contents of the data files that need archiving.

ReadWriteData first generates the datatype-independent data structures referencing the operational files. For efficiency reasons, *ReadWriteData* also updates some of the attributes in the data structures that will reference the archived data. After the references are updated (and finalized), *ReadWriteData* opens a socket connection to *MSwriter*. *ReadWriteData* then transmits the datatype-independent data

▼ Detailed Design

structure to *MSwriter* using the archive protocol. *ReadWriteData* receives the status from *MSwriter*. If no errors are encountered, *ReadWriteData* sends the contents of the operations files by sequentially reading the files into memory, computing a checksum, and transmitting the information to *MSwriter*. After the data are successfully transmitted to *MSwriter*, *ReadWriteData* inserts the data structures that reference the archived data into the archive database.

ReadWriteData has the following outputs: a data stream transmitted to *MSwriter* over the socket connection, the referencing database records written to the archive database (after *MSwriter* reports success), and a status value reported to *Archive* along with a text string describing any problems encountered during the operation of *ReadWriteData*.

Control

ReadWriteData is initiated as a function call from *Archive*. The process terminates when an error is encountered or when archiving of an interval is complete.

Interfaces

ReadWriteData has three primary interfaces. It interfaces with *Archive* through the argument list in the function call, it communicates with *MSwriter* using the archive protocol discussed in “Data Flow Model” on page 22, and it communicates with the database using the GDI.

Error States

As in *Archive*, error handling is critical in *ReadWriteData*. The most likely errors occur when reading from the filesystem, transmitting data to *MSwriter*, and inserting data structures in the archive database. The primary means of defensive programming is to use a transactional approach; that is, if any action reports an error, *ReadWriteData* aborts without corrupting any data. The transactional approach is assured by inserting the archive database records as the final step in *ReadWriteData*. The only mechanism by which the system can be corrupted is if the database

records are written prematurely or if the data are corrupt. If an unrecoverable error is encountered in *ReadWriteData*, then the database references are not written to the database.

Problems encountered in reading from the filesystem are typically the result of a configuration error or an error of upstream processing. The most probable system configuration errors are improperly set file permissions; that is, *ReadWriteData* is unable to open or read the file. If an error condition is encountered, *ReadWriteData* returns the error condition along with the filename that it was unable to open. *Archive* logs the problem, updates *interval.state* to **FAILED**, and then aborts processing. Another failure occurs if the file size is different from its expected size. In this case *ReadWriteData* reports to *Archive* that it was unable to read the expected number of bytes and aborts processing. *Archive* again logs the problem, updates *interval.state* to **FAILED**, and aborts processing.

A second class of errors occur because of problems encountered during communication with *MSwriter*, with several possible causes. One type of problem occurs when *MSwriter* is not able to either write the files or make new directories, as requested by *ReadWriteData*. The probable causes for this are: (1) the file permissions are not set properly, (2) the physical device is out of space, or (3) there is a hardware failure on the mass storage device. When this type of problem occurs, *ReadWriteData* captures the cause of the failure and aborts by returning an error code and error condition to *Archive*. *Archive* then logs the problem, updates *interval.state* to **FAILED**, and aborts processing. Another primary cause of failure is faulty communications, such as a lost socket connection or failure in validating the checksum. When these error conditions are encountered, *ReadWriteData* reports the problems to *Archive*, which logs the problem, updates *interval.state* to **FAILED**, and aborts processing.

MSwriter

MSwriter is a light-weight application that services the mass storage filesystem. *MSwriter* reads the control and data frames that are transmitted over a socket connection from *ReadWriteData* and writes the data to the mass storage filesystem. *MSwriter* also reports error conditions (should any *MSwriter* action fail) and vali-

▼ Detailed Design

dates checksums. In addition, *MSwriter* acknowledges all frames with either an ACK or NAK (Acknowledgment or Negative Acknowledgment) in a communications frame.

Input/Processing/Output

MSwriter receives inputs from a socket connection. The inputs are a data stream and control frames transmitted from *ReadWriteData*. *MSwriter* acts on the control frame (for example, make directory), verifies all received data frames (via a checksum), and writes the data to the filesystem. All transmissions from *ReadWriteData* require a response from *MSwriter*. The output from *MSwriter* is the archived data file stored on the mass storage filesystem and an ACK (or NAK) transmitted to *ReadWriteData*.

Control

MSwriter is initiated as an *inetd* daemon process; that is, *MSwriter* initiates when *ReadWriteData* (or any other application) opens a socket connection to the appropriate port number, which is port 3888. The port number is assigned in a header file using the variable *Archiver_Port*. *MSwriter* is controlled by *ReadWriteData*; *ReadWriteData* sends *MSwriter* commands, which *MSwriter* executes.

Interfaces

MSwriter interfaces with *ReadWriteData* using the archive protocol defined in "Flow of Internal Data Exchange (Archive Protocol)" on page 26. *MSwriter* interfaces with the mass store filesystem through standard UNIX commands.

Error States

MSwriter's role in error handling is to trap the errors and report them back to *ReadWriteData*.

DATABASE DESCRIPTION

The primary role of the Archiving Subsystem is to copy data files and their referencing database records from the operations database and filesystem to the archive database and mass store filesystem. *Archive* reads from the operations database via the GDI and writes to the archive database, also using the GDI. Because of the core role of the database in the operations of the Archiving Subsystem, it is appropriate to use the database for internal tracking and even for storing some archiving parameters.

Database Design

In addition to reading database records referencing data files that are to be archived and writing the corresponding archive database records, *Archive* uses a database for identifying, queuing, and tracking the data archiving process. *Archive* also uses the database to store some configuration information. Some information regarding particular datatypes, such as the database table name encapsulating the data, is stored in the database.

The entity-relationship diagram of the schema is shown in Figure 10. The diagram shows that the *datatype* attribute in the **arch_data_type** table maps to the *class* and *name* of the **interval** table. The **interval** table relates to multiple data tables, and the specific table is determined by **arch_data_type.table_name**. The **interval** table mapping is also somewhat dependent on the table name. For the **msgdisc** table, **msgdisc.itime** must be between **interval.time** and **interval.endtime** for **interval.class** = MSG and **interval.name** = MSG, or **msgdisc.idate** must be between **interval.time** and **interval.endtime** for **interval.class** = AUXARC and **interval.name** = AUXNET. For the **fileproduct** table, the relationship to the **interval** table is somewhat more complex. The **interval.class** and **interval.name** relate to a specific value of **fpdescription.prodtype**. The **fpdescription.typeid** links with the **fileproduct.typeid**. The *time* and *endtime* of the **interval** table relates with the *time* and *endtime* of the **fileproduct** table. The relation is **interval.time** ≤ **fileproduct.time** and **interval.endtime** ≥ **fileproduct.endtime**.

▼ Detailed Design

When the `arch_data_type.table_name` attribute is `wfdisc`, there are additional entity relationships. In this case, the archiving can have finer groupings; the waveform archiving can be queued by station and channel. To do this a station and channel must map to the `interval` table. The table `chan_groups` handles this mapping; it links the `class` and `name` in the `interval` table to the `sta` and `chan` in the `wfdisc` table. This linkage allows archiving to be configured to have, for example, all of the primary stations archived together by having all of the available `sta` and `chan` pairs map to one `name` and `class` pair. Similarly, each `sta/chan` pair can be archived separately by having each `sta/chan` pair map to an individual `class/name`. Every station or array is archived independently; that is, the `class` is set to primary archiving, and the `name` is set to the array name [IDC5.1.1Rev2]. Each element in the array (`sta/chan` pair) maps to the `name` and `class` of the array.

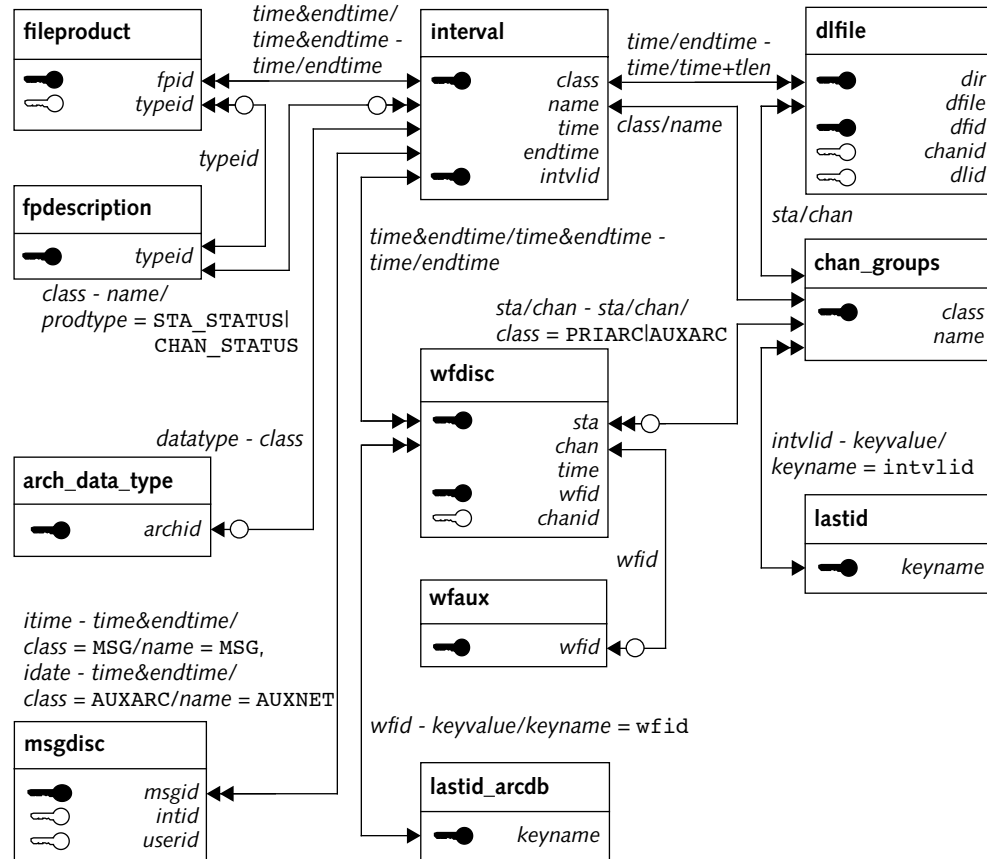


FIGURE 10. RELATIONSHIPS BETWEEN TABLES USED FOR ARCHIVING SUBSYSTEM

Database Schema

Table 9 shows the usage of database tables by the Archiving Subsystem. For each table used, the third column shows the purpose for reading or writing each attribute. The Action column shows if the table is read/written and in which database instance (operations or archive) the action is taken.

▼ Detailed Design

TABLE 9: DATABASE USAGE BY ARCHIVING SUBSYSTEM

Table	Action	Usage of Attributes
arch_data_type	ops read	<ul style="list-style-type: none"> • <i>table_name</i> indicates the database table storing the <i>dir</i>, <i>dfile</i>, <i>foff</i>, and <i>number of bytes</i> of the data to be archived. • <i>merge_adjacent</i> indicates that time-continuous records are to be combined into one longer segment (only applicable if <i>table_name</i> is wfdisc). • <i>data_type</i> specifies the type of data being archived and maps to the <i>class</i> attribute in the interval table.
chan_groups	ops read	<ul style="list-style-type: none"> • joins <i>sta</i> and <i>chan</i> from wfdisc to the <i>name</i> and <i>class</i> of interval.
dfile	ops read	<ul style="list-style-type: none"> • <i>time</i> and <i>endtime</i> are used to define the archiving interval (<i>time</i> and <i>endtime</i> in the interval table).
fileproduct	ops read, arch write	<ul style="list-style-type: none"> • <i>typeid</i>, <i>time</i>, and <i>endtime</i> define the processing intervals. • <i>dir</i>, <i>dfile</i>, <i>foff</i>, and <i>dsize</i> are used to locate the data being archived (<i>dir</i>, <i>dfile</i>, and <i>foff</i> are modified in archiving).
fpdescription	ops read	<ul style="list-style-type: none"> • <i>prodtype</i> identifies the type of product and is used to identify the row from a specific <i>name</i> and <i>class</i> in the interval table. • <i>typeid</i> is used to link the interval row to the fileproduct row (for a particular <i>prodtype</i>).
interval	ops read/ write	<ul style="list-style-type: none"> • <i>class</i> and <i>name</i> are used to identify and queue the archived elements. • <i>time</i> and <i>endtime</i> give the boundaries for the processing interval. • <i>state</i> is used for tracking and queuing the processing interval.
lastid	ops read/ write	<ul style="list-style-type: none"> • <i>keyvalue</i> is used for setting <i>intvlid</i>.
lastid_arcdb	arch read/ write	<ul style="list-style-type: none"> • <i>wfid</i> is used by the <i>Archive</i> application as a unique waveform identifier.

TABLE 9: DATABASE USAGE BY ARCHIVING SUBSYSTEM (CONTINUED)

Table	Action	Usage of Attributes
msgdisc	ops read, arch write	<ul style="list-style-type: none">• <i>dir</i>, <i>dfile</i>, <i>fileoff</i>, and <i>filesize</i> are used to locate the data to be archived.• <i>foff</i> and <i>mfoff</i> are modified in the archive processing.• <i>itime</i> is used to define the processing interval (<i>itime</i> is between interval.time and interval.endtime).
wfauX	ops read, arch write	<ul style="list-style-type: none">• <i>length</i> indicates the waveform length in bytes being archived.
wfdisc	ops read, arch write	<ul style="list-style-type: none">• <i>sta</i> and <i>chan</i> define the data being archived.• <i>dir</i>, <i>dfile</i>, <i>foff</i>, <i>nsamp</i>, and <i>datatype</i> define the data that are to be archived or have been written to the archive.

Chapter 4: Requirements

This chapter describes the requirements of Archiving Subsystem and includes the following topics:

- Introduction
- General Requirements
- Functional Requirements
- System Requirements
- Requirements Traceability

Chapter 4: Requirements

INTRODUCTION

The requirements of the Archiving Subsystem can be categorized as general, functional, or system requirements. General requirements are nonfunctional aspects of the Archiving Subsystem. These requirements express goals, design objectives, and similar constraints that are qualitative properties of the system. The degree to which these requirements are actually met can only be judged qualitatively. Functional requirements describe what the Archiving Subsystem is to do and how it is to do it. System requirements pertain to general constraints, such as compatibility with other IDC subsystems, use of recognized standards for formats and protocols, and incorporation of standard subprogram libraries.

GENERAL REQUIREMENTS

The Archiving Subsystem shall meet the following general requirements:

1. The archive destination shall be a target identified by a database table with attributes that express directory, filename, file position offset, and size.
2. The Archive Subsystem shall be capable of running unattended under normal circumstances.

FUNCTIONAL REQUIREMENTS

The requirements described in this section are categorized by function.

Generic Functional Requirements

3. The Archiving Subsystem shall be able to archive data referenced by database tables (that is, flat files). Each table will have attributes expressing directory, filename, file position offset, and size. Currently, the following tables will be considered: **wfdisc**, **msgdisc**, and **fileproduct**.
4. The Archiving Subsystem shall be configurable to group files together based on attributes in the database tables.
5. The Archiving Subsystem shall be configured to support an archiving schedule for different datatypes based on the delivery schedule of each datatype.
6. The Archiving Subsystem shall be able to archive “late arriving” data (that is, data that have arrived after this type of data should have arrived). Archiving late data will be handled automatically, and these data will be grouped with either the previously archived data or with other late arriving data.
7. The Archiving Subsystem shall provide a mechanism to exclude certain subtypes of data from being archived. The exclusion will be made based on database attributes.

User Interface

The Archiving Subsystem is required to interface with users as follows:

8. The Archiving Subsystem shall report on the *status* of the archiving processes.
9. The System Monitoring Subsystem software shall provide an “alert” mechanism to notify operators of a lack of storage space or other critical archiving related system problems.

▼ Requirements

10. The Archiving Subsystem shall provide a mechanism to selectively stop and start processes.
11. The Archiving Subsystem shall have the ability to use customized time constraints.
12. The Archiving Subsystem shall be able to work in two modes. In one mode, the input and output database table names will be the same. In the second mode, the input and output database table names will be different.
13. The Archiving System shall have the capability to optionally delete the original data file after it has been successfully archived. If the file is deleted, the reference(s) to that file in the input database table may be optionally deleted.
14. For time-series data, the Archiving System shall support organizing the data by time slices and/or by station.

Exception Handling and Recovery Procedures

The Archiving Subsystem is required to handle error conditions as follows:

15. The Archiving Subsystem shall not lose any data.
16. The Archiving Subsystem shall have the ability to migrate to another storage medium in the event of a major medium failure.
17. The Archiving Subsystem shall take precautions to ensure that the archived data are identical to the original data. These precautions will be limited to UNIX-level functions to verify that the input number of bytes equals the output number of bytes. Results of this test will be recorded in a log file.
18. The Archiving Subsystem shall provide a mechanism to verify that the bytes in the original data are identical to the bytes in the archived data.
19. The Archiving System shall preserve any compression and/or digital signatures stored in the original data.

20. The Archiving Subsystem shall preserve all identifiers (IDs) and load dates of the original data.
21. The Archiving Subsystem shall handle exceptional cases, for example, **wfdisc** records with zero or few samples.

SYSTEM REQUIREMENTS

The following are system requirements for the Archiving Subsystem:

22. The Archiving Subsystem shall use an ORACLE database.
23. The Archiving Subsystem shall use command line arguments to pass run-time parameters to the application software. These arguments will be provided in par files, and standard IDC software will be used for reading and parsing these files.

REQUIREMENTS TRACEABILITY

Tables 10 through 14 trace the requirements of the Archiving Subsystem to components and describe how the requirements are fulfilled.

TABLE 10: TRACEABILITY OF GENERAL REQUIREMENTS

	Requirement	How Fulfilled
1	The archive destination shall be a target identified by a database table with attributes that express directory, filename, file position offset, and size.	The archived files are referenced by the database tables on the archive database. The tables are wfdisc , msgdisc , and fileproduct .
2	The Archive Subsystem shall be capable of running unattended under normal circumstances.	The Archiving Subsystem is invoked from <i>cron</i> and has no operator inputs.

▼ Requirements

**TABLE 11: TRACEABILITY OF FUNCTIONAL REQUIREMENTS:
GENERIC FUNCTIONAL REQUIREMENTS**

	Requirement	How Fulfilled
3	The Archiving Subsystem shall be able to archive database referenced by database tables (that is, flat files). Each table will have attributes expressing directory, filename, file position offset, and size. Currently, the following tables will be considered: wfdisc , msgdisc , and fileproduct .	The Archiving Subsystem is capable of archiving data referenced by wfdisc , msgdisc , and fileproduct .
4	The Archiving Subsystem shall be configurable to group files together based on attributes in the database tables.	The chan_groups table enables the Archiving Subsystem to group waveforms by <i>sta-chan</i> . Other groupings can be configured by modifying the interval creation queries, which are stored as run-time parameters.
5	The Archiving Subsystem shall be configured to support an archiving schedule for different datatypes based on the delivery schedule of each datatype.	The queuing rules are defined in SQL statements stored as datatype-specific parameters. In addition, each instance of <i>Archive</i> can have its own timing as defined in the <i>cron</i> configuration. If multiple instances of <i>Archive</i> are running simultaneously, then multiple instances of <i>MSwriter</i> are initiated.
6	The Archiving Subsystem shall be able to archive “late arriving” data (that is, data that have arrived after this type of data should have arrived). Archiving late data will be handled automatically, and these data will be grouped with either the previously archived data or with other late arriving data.	This requirement has not been satisfied.
7	The Archiving Subsystem shall provide a mechanism to exclude certain subtypes of data from being archived. The exclusion will be made based on database attributes.	The datatype-specific parameters SQL provide a mechanism to exclude subtypes.

**TABLE 12: TRACEABILITY OF FUNCTIONAL REQUIREMENTS:
USER INTERFACE**

	Requirement	How Fulfilled
8	The Archiving Subsystem shall report on the status of the archiving processes.	The <i>status</i> is recorded both in log files and in the <i>interval.state</i> .
9	The System Monitoring Subsystem software shall provide an “alert” mechanism to notify operators of a lack of storage space or other critical archiving related system problems.	The Archiving Subsystem logs errors for lack of storage space and other system problems. The Archiving Subsystem also records the status of failed archiving intervals, which are displayed by the <i>WorkFlow</i> program.
10	The Archiving Subsystem shall provide a mechanism to selectively stop and start processes.	The Archiving Subsystem can be killed from the UNIX command line. Due to <i>Archive</i> ’s transactional processing model, no data are lost (though mass-storage space may be wasted).
11	The Archiving Subsystem shall have the ability to use customized time constraints.	The time constraints are datatype-specific and are stored in the configuration (par) files as part of the datatype-specific SQL statements.
12	The Archiving Subsystem shall be able to work in two modes. In one mode, the input and output database table names will be the same. In the second mode, the input and output database table names will be different.	By default, the Archiving Subsystem places the data in the standard tables (wfdisc , msgdisc , fileproduct). The outgoing database table name can be changed by a parameter. However, the table structure must match the expected table structure.
13	The Archiving System shall have the capability to optionally delete the original data file after it has been successfully archived. If the file is deleted, the reference(s) to that file in the input database table may be optionally deleted.	This requirement is not fulfilled. Purging is left to other process. For example, <i>DLMan</i> manages the recycling of disk loop files for the Continuous Data Subsystem CD-1.0.
14	For time-series data, the Archiving System shall support organizing the data by time slices and/or by station.	The chan_groups table allows the operator/user to define the station grouping.

▼ Requirements

TABLE 13: TRACEABILITY OF FUNCTIONAL REQUIREMENTS: EXCEPTION HANDLING AND RECOVERY PROCEDURES

	Requirement	How Fulfilled
15	The Archiving Subsystem shall not lose any data.	The archive protocol includes data size and checksums to ensure the correct number of bytes of data are transmitted and that the data have not been corrupted; in addition, the software is transactional.
16	The Archiving Subsystem shall have the ability to migrate to another storage medium in the event of a major medium failure.	This requirement is met by specifying a different destination machine and installing <i>MSwriter</i> on that machine.
17	The Archiving Subsystem shall take precautions to ensure that the archived data are identical to the original data. These precautions will be limited to UNIX-level functions to verify that the input number of bytes equals the output number of bytes. Results of this test will be recorded in a log file.	The archive protocol verifies that the number of received bytes is equal to the number of expected bytes. <i>Archive</i> only logs when the number of retries exceeds six (the Archiving Subsystem also aborts in that case).
18	The Archiving Subsystem shall provide a mechanism to verify that the bytes in the original data are identical to the bytes in the archived data.	The archive protocol provides a checksum during the archive transmission.

TABLE 13: TRACEABILITY OF FUNCTIONAL REQUIREMENTS: EXCEPTION HANDLING AND RECOVERY PROCEDURES (CONTINUED)

	Requirement	How Fulfilled
19	The Archiving System shall preserve any compression and/or digital signatures stored in the original data.	The Archiving Subsystem does not modify the data while archiving. Therefore, all compression and signatures are maintained.
20	The Archiving Subsystem shall preserve all identifiers (IDs) and load dates of the original data.	IDs are maintained for fileproducts and messages. With waveforms, this is not feasible, as adjacent waveform segments are merged. In addition, load dates are updated to indicate the time of archiving.
21	The Archiving Subsystem shall handle exceptional cases, for example, wfdisc records with zero or few samples.	This requirement is not fulfilled. A change request (CMRva00764) has been submitted to address this deficiency.

TABLE 14: TRACEABILITY OF FUNCTIONAL REQUIREMENTS: SYSTEM REQUIREMENTS

	Requirement	How Fulfilled
22	The Archiving Subsystem shall use an ORACLE database.	The Archiving Subsystem accesses the database using the GDI, which supports ORACLE.
23	The Archiving Subsystem shall use command line arguments to pass run-time parameters to the application software. These arguments will be provided in par files, and standard IDC software will be used for reading and parsing these files.	The Archiving Subsystem uses <i>libpar</i> for obtaining run-time parameters. <i>libpar</i> is an IDC global library and is compatible with parameter files.

References

The following sources supplement or are referenced in document:

- [DOD94a] Department of Defense, "Software Design Description," *Military Standard Software Development and Documentation*, MIL-STD-498, 1994.
- [DOD94b] Department of Defense, "Software Requirements Specification," *Military Standard Software Development and Documentation*, MIL-STD-498, 1994.
- [Gan79] Gane, C., and Sarson, T., *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.
- [IDC3.4.1Rev2] Science Applications International Corporation, Veridian Pacific-Sierra Research, *Formats and Protocols for Messages, Revision 2*, SAIC-00/3005, PSR-00/TN2829, 2000.
- [IDC5.1.1Rev2] Science Applications International Corporation, Veridian Pacific-Sierra Research, *Database Schema, Revision 2*, SAIC-00/3057, PSR-00/TN2830, 2000.
- [IDC7.3.1] Science Applications International Corporation, *Distributed Application Control System (DACS)*, SAIC-01/3001, 2001.
- [IDC7.4.2] Science Applications International Corporation, Pacific-Sierra Research, Inc., *Message Subsystem*, SAIC-98/3003, 1998.
- [IDC7.4.4] Science Applications International Corporation, *Subscription Subsystem*, SAIC-98/3001, 1998.

Appendix: Defining Fileproducts

This appendix describes how to configure the Archiving Subsystem to archive a regularly produced set of files (a fileproduct) using the fileproducts interface. New products that must be archived are usually added to the subsystem using the fileproducts interface.

Appendix: Defining Fileproducts

The Archiving Subsystem is designed to support user-defined fileproducts. The software design that underlies this capability is illustrated in the following procedure (with an example), which is used to define a fileproduct.

The procedure to define a fileproduct is to add a record describing the fileproduct to the **fpdescription** database table. The table is described in [IDC5.1.1Rev2]. The required contents are a unique identifier (*typeid*), the product type, a description of the product, the type of data, and the format of the data. With the description created, a **fileproduct** row referencing the filesystem object can be generated. The **fileproduct** row contains the directory (*dir*), filename (*dfile*), file offset (*foff*), and product size (*dsize*) as required by the Archiving Subsystem. In addition, it contains the *typeid*, which links to the **fpdescription** table, the time and end time of the object, and the station and channel codes if appropriate for the file object. The **fileproduct** rows can be added to the database by directly inserting the record, by using the Data Services CSCI library *libfileproduct*, or by using the Subscription Subsystem application *write_fp* [IDC7.4.4].

In addition to defining the fileproduct, the Archiving Subsystem must be configured to handle the new product. These steps are: 1) add a row to the **arch_data_type** table for the new data type, 2) define the datatype-specific parameters to create, queue, and read processing **intervals**, and 3) read the **fileproduct** database rows. As an example of this procedure, the steps involved in adding station capability reports to the archiving process follow.

First, add a row using the following SQL:

```
insert into arch_data_type values
( 3, 'STACAP', 'fileproduct', '-', '-', -1, -1, sysdate);
```

Then, define the datatype-specific configuration rules. For *datatype* STACAP, the interval creation rules are to generate the **interval** rows for the reports on a daily basis, back 30 days. The parameter value for this is as follows:

```
STACAP-interval_create="select distinct -1 intvldid, \
    'CAPARC' class, 'STACAP' name, \
    floor (fp.time/86400)*86400 time, \
    floor((fp.time+86400)/86400) *86400 endtime, \
    'NEW' state, to_date('{current_lddate}', \
    'YYYYMMDD HH24:MI:SS') moddate, \
    to_date('{current_lddate}', 'YYYYMMDD HH24:MI:SS') \
    lddate\
from fileproduct fp \
where fp.typeid in (select typeid from fpdescription \
    where prodtype in ('STA_STATUS', 'CHAN_STATUS')) \
and fp.time > {current_epoch} - 30 *86400 \
minus select distinct -1 intvldid, 'CAPARC' class, \
    'STACAP' name, i.time, i.endtime, 'NEW' state, \
    to_date('{current_lddate}', 'YYYYMMDD HH24:MI:SS') \
    moddate, \
    to_date('{current_lddate}', 'YYYYMMDD HH24:MI:SS') \
    lddate \
from $interval i \
where i.time > {current_epoch} -86400*30 \
and i.class='CAPARC' \
and i.name='STACAP'"
```

This query has two primary sections: the first section creates the **intervals** and the second section removes existing intervals. In addition, when *Archive* encounters {current_epoch}, it replaces it with the current (system) epoch time.

▼ Defining Fileproducts

The next parameter defines the queuing rules. For STACAP the intervals are queued 10 days after the reports are generated. Therefore, the query is fairly straightforward:

```
STACAP-interval_update="update interval\
  set state='QUEUED', \
    moddate=to_date('{current_lddate}', \
      'YYYYMMDD HH24:MI:SS') \
  where class='CAPARC' \
    and state in ('NEW', 'RETRY') \
    and time < {current_epoch} - 10*86400"
```

Two other parameterized database queries are defined: the query to read the queued intervals and the query to read the fileproduct records for that interval. For the example presented here, the read-interval query is in the parameter STACAP-interval_read; it reads all of the records with `interval.state='QUEUED'` for the `class 'CAPARC'`:

```
STACAP-interval_select="select * from $(INTERVAL) \
  where state='QUEUED' \
    and class='CAPARC' \
  order by time"
```

The final parameterized database query is in the parameter STACAP-read_data. This reads the **fileproduct** rows for the interval of interest. An example of the parameter value follows:

```
STACAP-read_data="select f.* from fileproduct f \
  where f.typeid in
    (select typeid from fpdescription \
      where prodtype in ('STA_STATUS', 'CHAN_STATUS')) \
    and f.time between %f and %f-.00001 \
  order by f.sta, f.time";
```

The first and second %f's in this parameterized query are replaced by the **interval.time** and **interval.endtime**, respectively. *Archive* performs this action using the C function, `sprintf`.

With the datatype-specific parameterized queries defined, there are a few other datatype-specific parameters that must be added:

```
STACAP-archive_directory  
STACAP-fileprefix  
STACAP-filepost
```

These parameter values are discussed in the *Archive* man page.

Glossary

Symbols

3-C

Three-component.

A

ACK/NAK

Acknowledgment or Negative Acknowledgment.

architecture

Organizational structure of a system or component.

architectural design

Collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

archive

Single file formed from multiple independent files for storage and backup purposes. Often compressed and encrypted.

archive (archival) database

Relational database that provides permanent storage of parametric and raw data.

archive server

Machine that hosts the mass storage device.

archive protocol

Protocol used for communications between *Archive* and *MSwriter*.

archive timestamp frame

Data structure that contains timestamps (start- and end-time), which is transmitted using the archive protocol.

array

Collection of sensors distributed over a finite area (usually in a cross or concentric pattern) and referred to as a single station.

ASCII

American Standard Code for Information Interchange. Standard, unformatted 256-character set of letters and numbers.

associated database tables

Database tables that contain the location of filesystem objects (for example, **wfdisc**, **msgdisc**, and **fileproduct**).

▼ Glossary

attribute

(1) Database column. (2) Characteristic of an item; specifically, a quantitative measure of a S/H/I arrival such as azimuth, slowness, period, or amplitude.

AutoDRM

Automatic Data Request Manager.

C**change request**

Procedure for reporting software problems or requesting upgrades.

CMR

Center for Monitoring Research.

command

Expression that can be input to a computer system to initiate an action or affect the execution of a computer program.

communications frame

Data structure that contains the operational status/next frame and is transmitted using the archive protocol.

component

(1) One dimension of a three-dimensional signal; (2) The vertically or horizontally oriented (north or east) sensor of a station used to measure the dimension; (3) One of the parts of a system; also referred to as a module or unit.

Comprehensive Nuclear-Test-Ban Treaty Organization

Treaty User group that consists of the Conference of States Parties (CSP), the Executive Council, and the Technical Secretariat.

Computer Software Component

Functionally or logically distinct part of a computer software configuration item, typically an aggregate of two or more software units.

Computer Software Configuration Item

Aggregation of software that is designated for configuration management and treated as a single entity in the configuration management process.

configuration

(1) (hardware) Arrangement of computer system or component as defined by the number, nature, and interconnection of its parts. (2) (software) Set of adjustable parameters, usually stored in files, for applications to use at run time.

configuration item

Aggregation of hardware, software, or both treated as a single entity in the configuration management process.

configuration management

Directing and surveying the functional and physical characteristics of a configuration item, controlling changes to those characteristics, and recording and reporting changes and implementation, and verifying compliance with requirements.

continuous waveform data

Waveform data that are transmitted to the IDC on a nominally continuous basis.

control flow

Sequence in which operations are performed during the execution of a computer program.

COTS

Commercial-Off-the-Shelf; terminology that designates products such as hardware or software that can be acquired from existing inventory and used without modification.

CPU

Central Processing Unit.

CSC

Computer Software Component.

CSCI

Computer Software Configuration Item.

CTBTO

Comprehensive Nuclear-Test-Ban Treaty Organization; Treaty User group that consists of the Conference of States Parties (CSP), the Executive Council, and the Technical Secretariat.

D**DACS**

Distributed Application Control System. This software supports inter-application message passing and process management.

daemon

Executable program that runs continuously without operator intervention. Usually, the system starts daemons during initialization. (Example: *cron*.)

data flow

Sequence in which data are transferred, used, and transformed during the execution of a computer program.

data frame

Data structure that contains some type of data and is transmitted using the CD-1.0, CD-1.1, or archive protocol.

detailed design

Refined and expanded version of the preliminary design of a system or component. This design is complete enough to be implemented.

disk loop

Storage device that continuously stores new waveform data while simultaneously deleting the oldest data on the device.

E**entity-relationship (E-R) diagram**

Diagram that depicts a set of entities and the logical relationships among them.

enumerated list

Construct that provides a convenient way to associate constant values with descriptive names.

▼ Glossary

epoch time

Number of seconds after January 1, 1970 00:00:00.0.

execute

Carry out an instruction, process, or computer program.

F**failure**

Inability of a system or component to perform its required functions within specified performance requirements.

file and data segment header frame

Data structure that contains the information necessary to define a file system object (dir, dfile, foff, and size). Data structures are transmitted using the archive protocol.

fileproduct

- (1) Object method for creating a database reference for any filesystem object.
- (2) Database table whose records describe files containing products.

filesystem

Named structure containing files in sub-directories. For example, UNIX can support many filesystems; each has a unique name and can be attached (or mounted) anywhere in the existing file structure.

filesystem object

Portion of a file on the UNIX file system that can be described by the file location and name (dir and dfile), an offset location within the file (foff), and its size (dsize).

G**GB**

Gigabyte. A measure of computer memory or disk space that is equal to 1,024 megabytes.

GDI

Generic Database Interface.

generic object

Construct used to hold and manipulate data. The type of object determines the data that it can contain. Also known as an object or GObj.

GUI

Graphical User Interface.

H**host**

Machine on a network that provides a service or information to other computers. Every networked computer has a hostname by which it is known on the network.

I**ID**

Identification; identifier.

IDC

International Data Centre.

IDC Operators

Technical staff that install, operate, and maintain the IDC systems and provide additional technical services to the individual States Parties.

IMS

International Monitoring System.

instance

Running computer program. An individual program may have multiple instances on one or more host computers.

IP

Internet protocol.

IP address

Internet Protocol address, for example: 140.162.1.27.

IPC

Interprocess communication. The messaging system by which applications communicate with each other through *libipc* common library functions. See *tuxshell*.

K**KB**

Kilobyte. 1,024 bytes.

M**mass storage device (mass store)**

Physical device that is capable of storing exceptional data volumes as part of the filesystem. Typically, this is a tape or disk jukebox that uses media such as Compact Disks (CD), Digital Versatile Disks (DVD), and Digital Linear Tape (DLT).

MB

Megabyte. 1,024 kilobytes.

O**object**

See generic object.

operations database

Relational database used by the operational system.

ORACLE

Vendor of the database management system used at the PIDC and IDC.

▼ Glossary

P

parameter (par) file

ASCII file containing values for parameters of a program. Par files are used to replace command line arguments. The files are formatted as a list of [*token* = *value*] strings.

pathname

Filesystem specification for a file's location.

PIDC

Prototype International Data Centre.

PIDC System Developers

Contractors and other organizations who are developing and testing components of the PIDC technology.

PL/SQL

Procedural Language for SQL.

process

Function or set of functions in an application that perform a task.

R

radionuclide

Pertaining to the technology for detecting radioactive debris from nuclear reactions.

RAM

Random Access Memory.

real time

Actual time during which something takes place.

recovery

Restoration of a system, program, database, or other system resource to a state in which it can perform required functions.

S

SAIC

Science Applications International Corporation.

schema

Database structure description.

script

Small executable program, written with UNIX and other related commands, that does not need to be compiled.

socket

Type of file used for network communication between processes.

socket connection

Method allowing a program on one machine to talk to a program on another machine over a Transmission Control Protocol/Internet Protocol (TCP/IP) connection.

software unit

Discrete set of software statements that implements a function; usually a sub-component of a CSC.

Solaris

Name of the operating system used on Sun Microsystems hardware.

SQL

Structured Query Language; a language for manipulating data in a relational database.

States Parties

Treaty user group who will operate their own or cooperative facilities, which may be NDCs.

station

Collection of one or more monitoring instruments. Stations can have either one sensor location (for example, BGCA) or a spatially distributed array of sensors (for example, ASAR).

station code (or ID)

(1) Code used to identify distinct stations. (2) Site code.

subsystem

Secondary or subordinate system within the larger system.

T**TCP/IP**

Transmission Control Protocol/Internet Protocol.

time, epoch

See epoch time.

transactional

Description of operations that are treated as a unit. If one of the operations fails, the set fails and the system rolls back to the state prior to the set of operations.

tuxshell

Process in the Distributed Processing CSCI used to execute and manage applications. See IPC.

U**UNIX**

Trade name of the operating system used by the Sun workstations.

V**version**

Initial release or re-release of a computer software component.

W**waveform**

Time-domain signal data from a sensor (the voltage output) where the voltage has been converted to a digital count (which is monotonic with the amplitude of the stimulus to which the sensor responds).

wfdisc

Waveform description record or table.

▼ Glossary

WorkFlow

Software that displays the progress of automated processing systems.

workstation

High-end, powerful desktop computer preferred for graphics and usually networked.

Index

A

arch_data_type 13, 43, 45

Archive 31

controlling 32

error states 33

I/O 31

interfaces 32

archive protocol 26

attribute usage 46

AutoDRM 20

C

chan_groups 13, 44, 45

COTS software requirements 7

Create Intervals 15

D

DACS 12

database 12, 43

schema overview 13

data exchange 26

data flow

external 22

internal 23

symbols iv

design

conceptual 10

database 43

dlfile 13, 45

E

entity-relationship

diagram 45

symbols v

F

file and data segment header frame

structure 29

fileproduct 13, 43, 45

fpdescription 13, 43, 45

G

GetData 36

controlling 37

error states 37

I/O 36

interfaces 37

global libraries 12

H

hardware requirements 6

▼ Index

I

interface
 with external users 20
 with internal users 20
 with operators 20
 with other IDC systems 19
interval 13, 43, 45
 IPC 12

L

lastid 14, 45
lastid_arcdb 13
 libraries, global 12

M

ManageInterval 34
 controlling 35
 error states 36
 I/O 35
 interfaces 35
 man pages iii
MergeData 37
 controlling 38
 I/O 37
 interfaces 38
 monitoring software 17
msgdisc 14, 43, 45
MSwriter 41
 controlling 42
 error states 42
 I/O 42
 interfaces 42

Q

Queue Intervals 16

R

ReadWriteData 39
 controlling 40
 error states 40
 I/O 39
 interfaces 40
 requirements
 COTS software 7
 exception handling and recovery 52, 56
 general 50, 53
 generic functional 51, 54
 hardware 6
 system 53, 57
 user interface 51, 55
Run Interval 17
 processing flow 18

S

schema 45

T

timestamp frame
 structure 28
 typographical conventions vi

W

wfaux 14
wfdisc 14
WorkFlow 10, 17, 20